Relational Data Imputation with Graph Neural Networks

Riccardo Cappuzzo
EURECOM
France
cappuzzo@eurecom.fr

Saravanan Thirumuruganathan
QCRI, HBKU
Qatar
sthirumuruganathan@hbku.edu.qa

Paolo Papotti
EURECOM
France
papotti@eurecom.fr

ABSTRACT

Performing data analysis over incomplete data produces biased results and sub-par performance. Imputation over relational datasets that contain both categorical and continuous variables is challenging. The challenges are accentuated when the missingness proportion of dataset is high, wherein a large fraction of the relation contain missing values, or if missing values occur in multiple attributes of a single tuple. In this paper, we propose GRIMP, a novel approach for imputation that tackles these challenges. GRIMP achieves high imputation accuracy through a combination of three novel ideas. First, it represents relational data as a heterogeneous graph, encoding sophisticated relationships between tuples, attributes and cell values. Second, it uses graph representation learning based on message passing to combine and aggregate the representations from appropriate neighborhoods. This allows GRIMP to leverage information from other cell values of the same tuple and that of similar tuples for imputation. Finally, it uses a self-supervised multi-task learning paradigm for training imputation models. In other words, GRIMP does not need any explicit training data as it uses the existing relational data, even when it has missing values. GRIMP trains an imputation model for each attribute using a two-stage approach consisting of a task agnostic section, where the parameters are shared across all attributes, and an attribute specific imputation model. Experiments over ten datasets and seven baselines show that GRIMP performs accurate imputation and provides new insights about the limitations of data imputation systems.

1 INTRODUCTION

Missing data is one of the most common data quality issues. Any analysis performed on the incomplete data would produce biased estimates leading to poor decision making. It can also affect the downstream applications, such as machine learning (ML), by reducing the amount and quality of complete training data. If the amount of missing data is minimal and the data is missing completely at random (MCAR), then a natural, if wasteful approach would be to ignore tuples with missing values during the analysis. In practice, many real-world datasets might contain too much missing data, or have systematic sources of missing values: here, omitting "dirty" tuples would result in biased data analysis.

Prior Work and their Limitations. There has been extensive work on data imputation. Discriminative models such as random forests or neighborhood methods often produce poor results in the presence of systematically missing data as they produce biased estimates that affect interpolation based approaches [52]. An alternate approach is to use (deep) generative models such as Generative Adversarial Networks (GANs) or denoising autoencoders to reconstruct missing values. These approaches make assumptions about data distributions. When these assumptions hold, they produce accurate imputation and better generalization. However, exemplars of both these classes of techniques have many limitations that minimize their effectiveness on relational data.

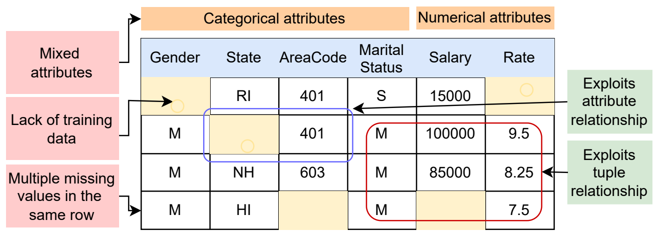


Figure 1: Example of data imputation challenges (left) and opportunities (right).

First, most of the prior work cannot handle mixed datasets containing both categorical and continuous attributes. Unfortunately, most of the relational datasets fall into this category, as depicted in Figure 1. A key challenge is that training a classifier requires multiple objectives, such as minimizing RMSE for continuous data and cross-entropy for categorical data. This is especially non-trivial for deep generative models, where poor training results in non-convergence or mode collapse [9, 52]. Second, many imputation models require a clean data subset for training. This might not always be possible when a large portion of the data has missing values, thus resulting in relatively poor performance. Third, popular approaches for multiple imputations (where a tuple might have multiple attributes with missing values) such as MissForest [46] or MICE [48] are iterative in nature. Specifically, they train m separate models for imputing m attributes wherein each model uses all attributes but one to impute the remaining attribute. This process is iterated over all features to produce a completely imputed dataset. A key issue with this approach is that each of the m models learns the imputation without sharing the commonalities. Finally, most of prior approaches do not take global relationships into account for imputation. Discriminative models are trained over individual tuples and used for imputing on individual tuples. However, it is beneficial to involve information beyond the single tuple – such as other similar tuples or meta-information such as functional dependencies that establish relationships between multiple attributes.

Two recent approaches leverage different kinds of information to improve imputation performance. AimNet [52] leverages the attention mechanism to learn the *relationships between attributes*, such as State and AreaCode in Figure 1. GINN [45] uses a graph convolutional layer within an autoencoder architecture. This allows the model to leverage *similar tuples*, as depicted in Figure 1 for the imputation of Salary in the last tuple. However, both approaches fall short by not completely leveraging all possible global information. Furthermore, GINN uses a similarity matrix for quantifying relationships between tuples, which requires quadratic time complexity and is unsuitable for large datasets.

Leveraging All Global Information. We introduce GRIMP (**G**raph embeddings for **R**elational data **IMP**utation), a novel graph based approach that tackles each of the aforementioned limitations.

Mixed Datasets. GRIMP represents the relational data as a concise heterogeneous graph allowing it to handle both categorical and numerical attributes. This graph allows one to capture global relationships between tuples and attributes. Additionally, the graph can easily be augmented to encode other domain specific information.

GRIMP formulates the imputation problem as a multi-task ML problem [13]. Specifically, GRIMP trains m models where the i-th model is used to impute attribute A_i. Imputation of categorical attributes is achieved using multi-class classification whereas imputation of numerical attributes is formulated as a regression problem.

No Training Data. GRIMP leverages the self-supervised learning paradigm so that it can be trained without the need for labeled data or a clean subset of data. This capability is especially appealing when the missingness proportion of the dataset is high.

Local and Global Information. Specifically, GRIMP uses graph neural networks (GNN) that produce lower-dimensional vector representation for any node by leveraging both structure and attributes of the graph [53]. This approach is *inductive*, which allows GRIMP to be used for imputation on tuples unseen during training. GNNs are trained through message-passing over multiple rounds, combining and aggregating the representations from relevant neighborhoods. This allows GRIMP to exploit local and global relationships, such as those between attributes and tuples.

A key innovation of GRIMP is a two step approach that avoids needless redundancies between the imputation models for each attribute. The first stage is a general *shared* section where the model parameters are shared across imputation models for each attribute. The second stage is a task specific setting that focuses on imputing an individual attribute. This shared approach allows sharing of relevant information across imputation models for different attributes resulting in superior accuracy. The imputation models for all attributes is trained end-to-end by using a dual loss function that combines the losses for categorical and numerical attributes.

Experimental Results. We evaluated GRIMP against six representative baselines, and we develop one additional baseline by extending one of them to handle external information. Experiments were executed over *ten* datasets with mixed data types. The experiments were conducted over missingness proportions up to 50%. GRIMP outperforms the baselines in almost all the cases and remains competitive with the best performing baseline in the remaining cases. We investigate the role of the key components of GRIMP and demonstrate that they are indispensable to provide improved results.

Contributions. The high level contribution of GRIMP is a graph based representation of relational data to encode local and global relationships. Once the graph is constructed, one could use graph representation learning to obtain node representations that provide good imputation. We note that GRIMP is agnostic to the specific GNN model used for representation learning. GRIMP handles heterogeneous data and multiple types of missingness patterns with arbitrary data distributions and missingness proportions as high as 50%. We summarize our four main contributions as follows.

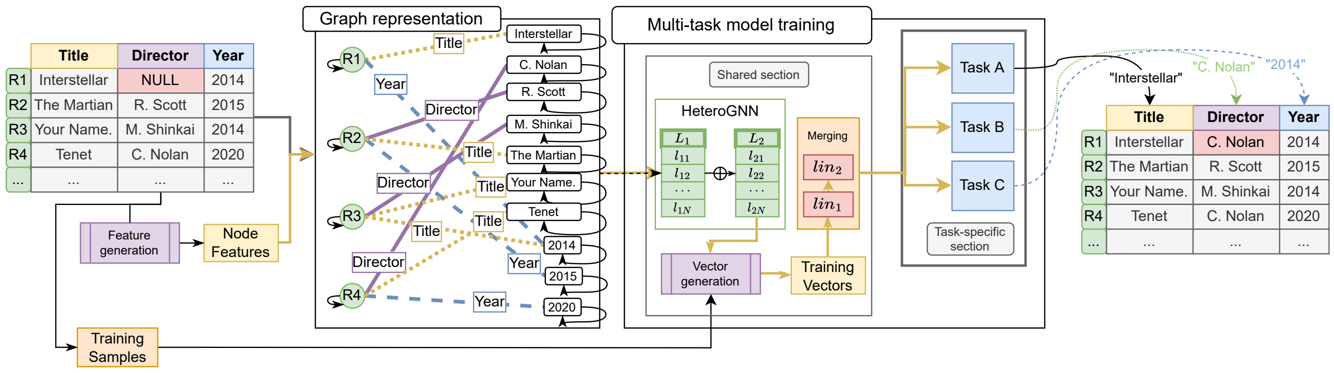
- We investigate the problem of multiple imputation over datasets with mixed data-types with significant amount of tuples with missing data.
- We propose GRIMP, which represents the relational data as a heterogeneous graph. It then uses graph representation learning for obtaining the node embeddings that can be used for imputation of individual attributes.
- GRIMP tackles the multiple imputation problem using the paradigm of multi-task learning. GRIMP uses a two stage approach – a shared task agnostic section and attribute specific imputation model based on attention.
- We run a systematic experimental campaign over ten datasets and seven baselines to show that (1) GRIMP can perform accurate imputation and (2) it can exploit external information in the form of functional dependencies to improve performance. We conclude with a section on error analysis, where we show that most imputation methods share weaknesses and that imputing rare values is the hardest part for all proposed solutions.

Paper Organization. We introduce the problem setup and terminology in Section 2. Section 3 describes the components of GRIMP such as graph construction, representation learning and multi-task imputation. We report the results of our experimental analysis in Section 4. We discuss the limits of imputation methods for relational data in Section 5. We describe the relevant prior work on imputation in Section 6. Finally, Section 7 discusses limits and opportunities of using deep learning for data imputation.

2 BACKGROUND

Let $\mathcal{D}$ be a relational dataset with n tuples $\{t_1, t_2, \dots, t_n\}$ and m attributes $\{A_1, A_2, \dots, A_m\}$ which contains missing values. Let $\mathcal{R}$ be the schema of $\mathcal{D}$. Each attribute $A_i \in \mathcal{R}$ could be either categorical or numerical. Let $C(\mathcal{R})$ be the set of attributes that are categorical. Let $N(\mathcal{R})$ be the set of attributes that are numerical. We denote the domain of attribute A_i as $Dom(A_i)$. Without loss of generality, let $Dom(A_i)$ for a categorical attribute be $\{1, 2, \dots, |A_i|\}$, where $|A_i|$ is cardinality of the domain of attribute A_i. We denote the value of attribute A_j in tuple t_i as $t_i[A_j]$. Missing values are represented by a special sentinel token $\varnothing$. We consider two variants of the dataset $\mathcal{D}$: let $\tilde{\mathcal{D}}$ be the imputed dataset where every entry with $t_i[A_j] = \varnothing$ is replaced with a value from $Dom(A_j)$, and let $\mathcal{D}^*$ be the ground truth version of $\tilde{\mathcal{D}}$.

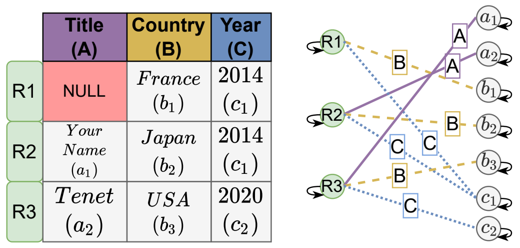
Data imputation is the task of processing the dataset $\mathcal{D}$, which contains missing or *erroneous* values, with the objective of correcting the errors on the basis of the context provided by the existing features. We assume that an orthogonal error detection procedure has been used to mark erroneous cells with $\varnothing$ [1, 36]. Intuitively, the goal of an imputation algorithm $\mathcal{A}$ is to output a dataset $\tilde{\mathcal{D}}$ where every entry with $t_i[A_j] = \varnothing$ is filled with the value $t_i^*[A_j]$ where the tuple t_i^* is the complete version of t_i from $\mathcal{D}^*$. The accuracy of the imputation is evaluated based on the type of the attribute. For numerical attributes, we use RMSE. The imputation of a categorical cell value is consider as accurate if $t_i[A_j] = t_i^*[A_j]$; the overall accuracy is simply the number of accurate imputations.



	Title	Director	Year
R1	Interstellar	NULL	2014
R2	The Martian	R. Scott	2015
R3	Your Name.	M. Shinkai	2014
R4	Tenet	C. Nolan	2020
...	...	...	...

	Title	Director	Year
R1	Interstellar	C. Nolan	2014
R2	The Martian	R. Scott	2015
R3	Your Name.	M. Shinkai	2014
R4	Tenet	C. Nolan	2020
...	...	...	...

Figure 2: Overview of the GRIMP **architecture. Given the dataset, its training samples and its graph are created and both are fed to the multi-task model. The shared and the task-specific sections in this model are jointly learned at training time. Every task emits imputation for missing values for a specific attribute.**



	Title (A)	Country (B)	Year (C)
R1	NULL	*France* (b_1)	2014 (c_1)
R2	*Your Name* (a_1)	*Japan* (b_2)	2014 (c_1)
R3	*Tenet* (a_2)	*USA* (b_3)	2020 (c_2)

Figure 3: Record ID nodes and value nodes are color-coded.

GRIMP's graph is a heterogeneous, quasi-bipartite graph which encodes the table's content and its structure. The graph has multiple node types to represent entities (tuples, and cells) and different edge types to represent relationships between them. Figure 3 shows a sample dataset on the left, together with the corresponding graph. In the graph, each tuple is assigned a RID node (record id nodes highlighted in green). Each unique value in the dataset (belonging to all attributes) is assigned a cell node (shown in grey). The RID and cell nodes are connected via a typed edge. The edge type is defined by the attribute of the cell, with the number of types being equal to the number of attributes in the table. Typed edges are color coded according to the respective attribute and are independent from the attribute labels. For example, tuple R1 is connected to value 'France' ('b_1') via a yellow edge with type 'Country' ('B'). Following the literature [33, 53], self-loops are added to the graph.

3 GRIMP

In this section, we introduce GRIMP (**G**raph embeddings for **R**elational data **IMP**utation), a generic data imputation framework.

3.1 System Architecture

An overview of GRIMP is illustrated in Figure 2.

Preprocessing. Given a input dirty (i.e., which includes missing values) table $\mathcal{D}$, GRIMP first runs a pre-processing step where it builds the graph and the training corpus. In the graph construction step (detailed in Section 3.2), GRIMP constructs a heterogeneous graph that encodes the table by generating a node for every row and every value, with typed edges linking such nodes. The training corpus is composed of copies of the table tuples with synthetically injected missing values that are split by attribute among the various tasks (Section 3.3).

Training. During the training procedure, GRIMP uses a GNN [27, 33] to refine the pre-trained node features by leveraging the structure of the graph. Given a node, its features are modified by combining them with those of its neighbors. Correspondingly, the features of the node's neighbors are also modified (Section 3.4). The computed embeddings are forwarded to the multi-task [49, 57] component that is responsible for the data imputation. This consists of two parts.

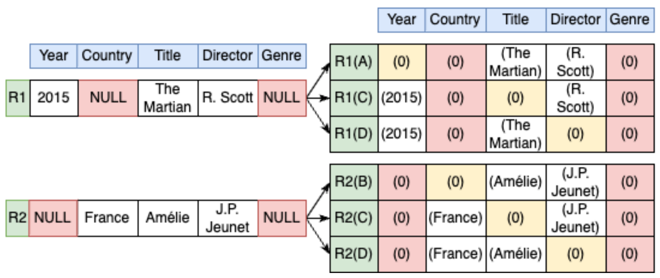
The first part uses hard-parameter sharing wherein the parameters of the GNN model are shared by all tasks. The second part focuses on imputing individual attributes. This contains "sub-models" whose weights are kept hidden from each other (Section 3.5). In other words, the learned parameter values are specific to the imputation of a given attribute. We dub these sub-models "tasks" and create one such task for each table attribute. Depending on the datatype, tasks are built as multi-class classifiers, or as regressors. We implement an attention structure in the tasks, which combines the vectors generated by the GNN with attribute-level information. Each task has its own loss function, depending on whether it is categorical or numerical, which is aggregated with that of other tasks to find the overall loss of the model (Section 3.6).

Imputation. After the training is complete, GRIMP performs imputation on the dirty data by selecting the appropriate value for the missing entries in the input table (Section 3.7).

3.2 Graph Construction

The first step is to construct a graph using the given dirty relational table $\mathcal{D}$. The graph is created by iterating over the dirty dataset row by row. A new node is added for each tuple and for each unique value in the tuple. A typed edge connects the tuple node to the value for that attribute. If a cell contains a missing value, no edges are added to the graph and the empty cell is ignored at this stage. Values that appear in multiple attributes are disambiguated so that each occurrence is connected exclusively to its attribute. If the same value v occurs in two separate attributes, the two occurrences are disambiguated by creating two nodes for v.

After graph creation, the node features are initialized. GRIMP represents each node as a low dimensional vector that can be used for various downstream tasks, including imputation. While



Year Country Title Director Genre
R1 2015 NULL The Martian R. Scott NULL
R2 NULL France Amélie J.P. Jeunet NULL
Year Country Title Director Genre
R1(A) (0) (0) (The Martian) (R. Scott) (0)
R1(C) (2015) (0) (0) (R. Scott) (0)
R1(D) (2015) (0) (The Martian) (0) (0)
R2(B) (0) (0) (Amélie) (J.P. Jeunet) (0)
R2(C) (0) (France) (0) (J.P. Jeunet) (0)
R2(D) (0) (France) (Amélie) (0) (0)

Figure 4: Example of training samples generated for two tuples with null values. Each tuple leads to a training sample with a new null (yellow cell) for each non-null value. Names (in parenthesis) denote that the object is an embedded representation of the original cell value.

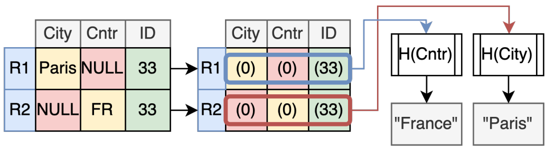


Figure 5: Example of a null value distribution that leads to the production of the same training vector.

we *learn* representations, it is possible to accelerate the learning through appropriate initialization. Numerical values are treated as strings and represented as nodes. To handle real numbers, we round them to a pre-defined number of decimal places (8 places by default). To avoid scale issues in the loss computations, numerical values are normalized before training the model, and then de-normalized before measuring the imputation accuracy.

3.3 Building the Training Corpus

GRIMP employs a representation of the tuples that is efficient and robust to noise. Every non-missing value is used for training the model by creating copies of the original tuple. Since the correct imputation for an injected missing value is known, we can use the model to do the imputation and measure a loss during the training. Each tuple is replicated as many times as there are attributes without missing values, with each copy being a different *training sample*.

For example, looking at Figure 4, tuple R1 is replicated three times, one time for each column with no missing values – first, by removing the value "2015" ($R_1(Year)$ = [∅, ∅, (The Martian), (R. Scott), ∅]), then by removing "The Martian" ($R_1(Title)$ = [2015, ∅, ∅, (R. Scott), ∅]), and finally by removing "R. Scott" ($R_1(Director)$ = [2015, ∅, (The Martian), ∅, ∅]).

This approach allows GRIMP to leverage every tuple (even if it contains a large fraction of "true" missing values) for generating training samples. As we describe later, each training vector is fed to a separate attribute specific imputation model. As a result, the same input fed to different tasks leads to different imputations. In general, if a tuple has K non missing attributes, it generates K synthetic tuples. Figure 4 shows the cases for K=2 ($R1$) and K=3 ($R2$). K does not depend on the size of the attribute domain and is bounded by the number of attributes.

The example in Figure 5 reports training samples for column "City" in row R1 and "Cntr" in row R2 (yellow background); both lead to the same training vector (middle part of the figure). If the model used a single classifier, such classifier would produce only one output, choosing either "France" or "Paris" as its imputation. Instead, by using multiple *independent* tasks, it feeds the same vector to different classifiers and obtain different guesses. The model has to handle this case with every tuple that contains more than one missing value. By designing the system in such a way that the occurrence is accounted for during training, we ensure resilience even in scenarios with high error fractions.

3.4 Graph Representation Learning

Once the graph and the training corpus are constructed, they can be used to learn node representations. The node features can be initialized randomly, with some pre-trained corpus, or with a local embedding model. However, it is possible to refine them so as to achieve accurate results in the downstream task of imputation. While there are many approaches for learning such representations, GRIMP uses Graph Neural Networks (GNNs) [25, 53]. GNN based representations are inductive, i.e., try to learn generic patterns from the observed training data (such as relationships between nodes or based on a neighborhood), which allows them to be used for imputing tuples that were unseen during training.

GNN takes the graph as input along with a set of node features X, then aggregates the features of each node with its neighbors via a message passing operation [22]. The features are then transformed by a non-linear function. Intuitively, every node representation encodes information relative to the node itself and to its direct neighbors. Similar nodes (either in features or neighborhoods) are then placed close to each other in the vector space of the embeddings. By adding more layers on top of each other, it becomes possible to propagate a node's features to longer distances. The output of the stack of layers contains the distributed representations of the graph nodes, which is then used to perform tasks such as imputation. GRIMP implements a heterogeneous spatial-based convolutional GNN that enables the use of different GNN models with different edge types. The architecture is described in Section 3.5.

Pre-Trained Features. GNNs combine the features a node's neighbors with the node's own features to produce a merged representation of the node itself. Initial feature representations for tuples and cell values can either be randomly generated, or prepared by relying on external methods. We propose two strategies for initialization.

The first solution relies on *pre-trained embeddings* [7]. Cell values are passed to the pre-trained embeddings algorithm, which returns a vector representation for each value. Then, the vector representation of a tuple is prepared by averaging the vector representations of the cell values in it. The vector representation of each attribute is prepared by averaging the vectors of the values in the attribute. These vectors are used in matrix Q in the attention layer.

Alternatively, one could use a framework for learning relational embeddings, such as EmbDI, to generate *local embeddings* from the original table [11]. We extend the EmbDI graph to better model null values by introducing a new edge type which represents "possible" imputation values. For example, given a missing value $t_i[A_j]$ in tuple t_i and attribute A_j, the graph has new edges connecting t_i's node to every value in the domain of A_j. By introducing these edges, the graph is aware that a missing value can take any of the values in the attribute domain. To model the fact that these edges are "possible", each edge e_{ix} connecting tuple t_i to cell value $x = t_i[A_j]$ is weighted proportionally to the frequency of x in attribute A_j, with more frequent values having higher weight.

3.5 Multi-Task Learning Component

The goal of graph representation learning to is to learn node representations to be used for some downstream task. The downstream task for GRIMP is data imputation.

One approach is to use a single classifier for all attributes. However, this directions leads to several problems. First, it has a very large label space, thus the classifier is error prone. Second, it is expensive as it requires more parameters and thereby more training data. Finally, this might often not be necessary in the case of well designed schemas where leakage of values from one attribute to other is uncommon.

A natural alternative is to have one classifier for each attribute. Consider a naïve translation of this idea. Given a tuple t, let the cell values be $\{t[A_1], t[A_2], \dots, t[A_m]\}\}$. Of course, some of these values could be missing. We could train an imputation model C_i that takes values $\{t[A_1], t[A_2], \dots, t[A_{i-1}], t[A_{i+1}], \dots, t[A_m]\}\}$ and seeks to predict $t[A_i]$. There are two issues with this approach. First, learning representations for each classifier C_i is wasteful. We tackle this through the multi-task learning wherein GRIMP learns a representation that simultaneously allows all imputation models $\{C_1, \dots, C_m\}$ to have high accuracy. Second, it seems wasteful to *independently* learn the imputation models. Intuitively, every imputation model learns a pattern in the data that can be useful to other models. For example, it is possible that the imputation models for two attributes A_i and A_j could use some common information. Naively training models C_i and C_j without leveraging this commonality results in an increase of model parameters, thus an increase in training time and potentially lower accuracy.

GRIMP's imputation module employs a multi-task learning (MTL) architecture for performing imputation over each attribute in the dirty dataset. This is achieved through two sections – shared and task-specific. In the shared section, the parameters are shared among all tasks through *hard parameter sharing* [57]. Intuitively, hard parameter sharing learns a common representation space for all tasks. In the task-specific section, each specific imputation task corresponds to an attribute. This architecture has a number of advantages over a simpler model that is trained on the full domain of the table. As discussed above, each task is trained on its attribute domain, which is much smaller compared to the full domain of the table, thus increasing imputation accuracy and reducing training time. Moreover, the same input vector can be used by different imputation tasks. Each task can produce different outputs given the same input based on the target attribute to be imputed. This is useful when a tuple contains multiple missing values, and therefore receives the same vector in the testing step. Finally, each task can use a different loss function and it allows us to transparently support different attribute types, such as categorical and numerical.

The structure of the GRIMP's MTL method is illustrated in Figure 2. It takes as input the graph and the set of training samples (Section 3.3). The shared section includes the GNN and an additional linear layer, which recombines the vectors produced by the GNN. The task-specific section has one task for each attribute. Depending on the attribute datatype, the output of the task varies accordingly. We describe the individual sub-components of the MTL next.

Shared Layer and GNN. The shared layer is composed of a Heterogeneous GNN and a merging step (depicted in orange in Figure 2). The Heterogeneous GNN is a data structure that can combine multiple GNN models in each layer, before forwarding them to then next layer. Our HeteroGNN block is composed of two layers, each layer l_{ij} being a GNN that handles a single column. Each submodule can use a different GNN architecture (e.g., l_{11} using GCN, l_{12} uses GraphSAGE and so on), however in this work we employ GraphSAGE [27] for all submodules. Additionally, it is also possible to use a different operator in each sub-module by leveraging the typed edges. A layer $L_i \in \{1, 2\}$ comprises N sub-modules (where N is the number of columns in the original dataset), where a sub-module is defined as $l_{ij}\ \forall i \in \{1, 2\} \land \forall j \in [1, 2, \dots, N]$ and each column in the starting dataset is assigned a sub-module. Each sub-module l_{ij} performs its convolution exclusively on nodes connected by edges of the type it pertains to (e.g., values belonging to column 2 are described by sub-modules $l_{12}, l_{22}, \dots, l_{n_{layers}2}$). As the final parameters are shared among all tasks, this set of operations is part of the "shared layer" in the GRIMP system.

This module takes as input the graph and combines it with a set of pre-trained features for each node in each of the two convolutional layers. Each layer combines the pre-trained features with the locality features given by the graph's adjacency matrix. The output of the overall network is the output of the final layer, which corresponds to the updated representation for each node. Between the convolutional layers, a pooling component combines the node representations. These vectors are produced by the final layer and used in the *vector generation* procedure. These could be then be used in the next steps of the procedure for updating the representation. The representation of a vector v after layer k is modeled as follows:

$$h_v^{(k)} = \sigma(\gamma(\mathbf{W}^{(k,i)} \cdot f_k^{(i)}(h_v^{(k-1)}, \{h_u^{(k-1)}, \forall u \in S_{\mathcal{N}(v)}\})\, \forall i \in [1, N])) \tag{1}$$

The formula describes the output of each layer of the GNN. Specifically, function $f_k^{(i)}(\cdot)$ combines the features of vector v generated in the previous epoch with those of all its neighbors $u \in S_{\mathcal{N}(v)}$; $\mathbf{W}^{(k,i)}$ is the matrix of learnable weights which is updated during each training epoch; γ is a function that aggregates the modified weights produced by each sub-module GNN_i, and σ is a nonlinearity.

By default, GRIMP uses two layers for the GNN. The trainable weights are not shared among sub-modules, which allows some independence between each column while modeling each node's feature representation. The "merging step" box shown in Figure 2 is implemented with two linear layers as a further pooling step. The intuition is to not use GNN embeddings directly, but further aggregate them using an intermediate step before the downstream task. Once the representation of each node is available, the training samples from the previous step are converted into *training vectors*. For each training sample, its values are replaced by their GNN-generated vector. Finally, the training vectors are forwarded to the task-specific layer for the imputation operation.

MTL Task-specific Layers. The task-specific layer is where the imputation operation is carried out. Each attribute in the dataset is assigned a *task*, whose characteristics depend on the type of attribute. A categorical attribute gets as task a multi-class classifier whose domain is the same as the domain of the attribute. A numeric attribute gets a regressor with a single output as task.

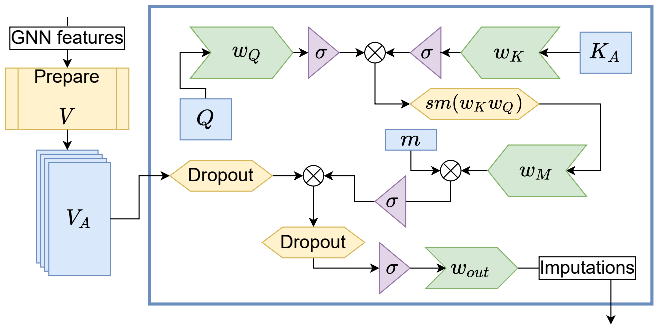


Figure 6: Classification task in the multi-task model.

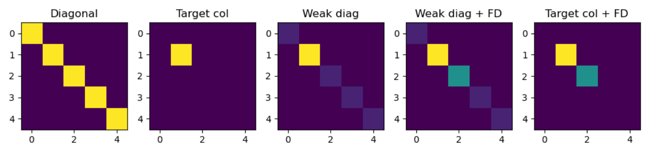


Figure 7: Different variants of matrix K in the task for attribute 2, with an FD between attribute 2 and 3.

During training, each task takes as input a subset of the training vectors relative to its attribute. From Figure 3, the training samples for tuple $R1$ are based on attributes *Country* and *Year*: the task for *Country* gets access to sample $t_{11} = [\varnothing, \varnothing, 2014]$, while the task for *Year* uses sample $t_{12} = [\varnothing, France, \varnothing]$. Matrix V is the collection of all training vectors for all tuples and all attributes: this matrix is used later in the attention step. As different attributes have different numbers of missing values, each task might take a different number of training vectors. For example, in Figure 3, attribute *Title* has two vectors, while both the other attributes have three. The additional information contained therein can be employed by the attributes that have fewer training vectors to work with.

In GRIMP, tasks can be implemented using linear layers that rely exclusively on the training vectors, or through an attention layer that combines attribute-level information with the training vectors. While the first solution is faster to train, the latter has better results. Linear tasks provide a simple architecture for keeping some parameters isolated from all other tasks, so that the classification (or regression) objective in each task is not as influenced by the other tasks. Shallow architectures (up to three linear layers) are enough to obtain good classification results.

Attention Structures. Inspired by the extensive work on attention mechanisms, we extend GRIMP's classifier architecture with an attention structure. We optimize for our scenario the architecture introduced in AimNet [52]. This attention structure is reserved to the task-specific layer, and is not used in the shared layer. Empirically, we observed that this approach outperforms an architecture that includes the attention structure on both shared and task-specific layers. Intuitively, this improvement can be explained based on relationships between attributes. For example, two attributes A_i and A_j could have a functional dependency (FD) wherein the value of A_i completely determines the value of A_j. Hence, using an attention structure in the task for A_j allows it to give higher weight to the value of A_i. This approach is extensible to more complex relationships such as FDs involving multiple attributes.

GRIMP's task attention structure is shown in Figure 6. The attention matrices Q, K, and V_A are shown in light blue, along with the pooling vector $\mathbf{m}$. For this example, we focus on the input to a single task H_A for attribute A. Each task receives a set of specific parameters with the same suffix as the task; for example, K_A refers to the matrix K that is built for task H_A.

Given the collection of node vectors prepared by the GNN V (with dimensions $N \times C \times D$), and the index vectors prepared as shown in Section 3.3 ("GNN Features" in Figure 6), matrix V_A is a subset of V which contains only the N_A training vectors relative to task A. It has shape $N_A \times C \times D$ where N_A is the number of training vectors for attribute A, C is the number of columns, and D is the number of dimensions of the vector for each node, i.e., the number of output dimensions of the shared layer.

Matrix Q_A has shape $C \times D$ and contains the pre-trained vectors of each column in the dataset; matrix Q_A holds the attribute information that must be combined with the information produced in the shared layer. The content of the matrix Q_i is the same for all attributes $i \in [1, \ldots, C]$ when the tasks are built, but each task H_i modifies its own Q_i independently. K_A is a binary matrix that is used for selecting only a set of the columns that each task should work with, assigning each column a given weight. K can be used to encode additional, external information by modifying the weight that is assigned to each column. If, for example, functional dependencies that link multiple columns are known, it is possible to "highlight" those columns in the matrix so that the model give them a higher weight during training.

For building K, we test four strategies, which differ in the values found on the diagonal of the matrix, as depicted in Figure 7. In the **Diagonal** variant, all columns have equal weight; with **Target column** all columns except the task's column are ignored; with **Weak diagonal** the target column has the highest weight, but other columns are still considered with a lower importance; **Weak diagonal + FD** implements the Weak diagonal strategy, but assigns columns involved in an FD a higher weight. Finally, $\mathbf{m}$ is a vector of size $1 \times C$ which contains 1 values. This vector pools the result of the multiplication of K_A and Q_A; this step is done to select the attributes that should be attended to the most by the current task.

Each task H_i is trained by passing V_i, a subset of the training vectors V, which is passed to the attention layer. In each task H_i, matrix Q_i and K_i are multiplied before being pooled by vector $\mathbf{m}$. After multiplying the result with matrix V_i, the final matrix passes through a linear layer whose output size is either equal to the cardinality of the domain of the task's attribute if the column is categorical, or one-dimensional if the column is numerical.

3.6 Training Procedure

The training procedure of GRIMP is summarized in Algorithm 1. The first step of our training revolves around the construction of the graph and of the training samples. Once the training samples are prepared, the model is trained iteratively over a given number of epochs. The training duration, expressed as number of epochs required, varies dataset by dataset, so 20% of the training samples is held out in a validation step to implement early stopping policies. We remove all edges incident in the validation step from the graph representation before training.

Algorithm 1 Pseudocode of the GRIMP pipeline.

```
Input: Table $\mathcal{D}$
NormalizeNumericalAttributes (table $\mathcal{D}$)
$G$ = GenerateGraph (table $T$)
$S$ = GenerateTrainingSamples (table $\mathcal{D}$)
$\mathcal{H}$ = BuildMultiTasks (table $\mathcal{D}$)
for epoch $E$ in $n_{epochs}$ do
    $h$ = GNN(G) {Generate node embedding using the GNN.}
    $V$ = BuildTrainingVectors($h$, $S$)
    $h_{shared}$ = SharedLayer($h$, $V$) {Feed the node embeddings to
    the shared layer of the multitask classifier.}
    for task $H_i$ in $\mathcal{H}$ do
        $h_i$ = TaskLayer($H_i$, $V$, $h_{shared}$)
        if type($H_i$) is numerical then
            $loss_i$ = RMSE($h_i$, $S$)
        else
            if type($H_i$) is categorical then
                $loss_i$ = CrossEntropy($h_i$, $S$)
    $loss_E = \sum_i^{n_{tasks}} loss_i$
```

Loss Function. In each epoch, all tasks measure their own loss independently of the others and according to their type: categorical attributes use *Cross Entropy* loss or *Focal* loss, while numerical attributes use MSE. Numerical values are normalized before the training starts, so that their MSE is comparable in magnitude to the Cross Entropy loss measured for categorical variables. This step allows to combine the loss of each task in a total loss (for the multi-task setting) with a simple summation.

3.7 Imputing the Missing Values

Imputation is done by preparing the vector representation of each dirty tuple in the table that are fed to the trained model. In our running example, tuple $R1$ becomes vector $t_1 = [\varnothing, France, 2014]$, and imputation is carried out by task H_A. For categorical values, imputed values are chosen by selecting the value with the highest likelihood, while numerical variables are imputed by taking the output of the numeric task, then de-normalizing it to generate the imputation. When imputing a tuple with a missing value for attribute A_i, GRIMP only considers the candidate values from $Dom(A_i)$ which is the set of all values from attribute A_i.

4 EXPERIMENTS

We test GRIMP against seven baselines across ten datasets. In all experiments, we start from the assumption that we do not have access to the ground truth during the training procedure. To test the final imputation accuracy, we introduce errors in datasets that contain no missing values, thus producing "dirty datasets" for which we have the ground truth. Code and datasets are available online [10]. More experimental results and details are in the technical report [12].

4.1 Experimental Setup

Datasets. We employ mixed-type datasets to evaluate imputation methods for categorical data. **Adult**, **Australian**, **Contraceptive**, **Credit**, **Flare**, **Mammogram**, **Thoracic**, and **Tic-Tac-Toe** are from the UCI repository [20], **IMDB** is a popular dataset [11], **Tax** is a synthetic dataset for testing data repair algorithms based on FDs [6] and denial constraints [14]. The datasets show a large variety of setting, with the size of the dataset domain varying between 5 and 9829 distinct values and the number of columns between 6 and 17.

Table 1 reports statistics about the datasets and our model. We divide the statistics in three main parts. The first one reports traditional measures (number of rows, categorical and numerical columns, unique values, FDs). The second part includes several metrics that we use for our error analysis in the next section. Finally, the last part reports information about GRIMP's parameters. We describe GRIMP's parameters next and the metrics in Section 5.

Algorithms. We use as baselines MissForest [46] (MISF), Holoclean/AimNet [52] (HOLO), TURL [19], EmbDI [11] (EMBDI-MC), and DataWig [5] (DWIG). We also tested a baseline based on link prediction that is not reported because of sub-par results. It uses link prediction within the graph, i.e., for a missing value, we predict if there is an edge between its node (attribute/row pair) and every value in the active domain. However, the graph topology is not rich enough to lead to acceptable results with this approach. We use the default parameters for all systems. We tested different combinations of parameters to set the default for GRIMP and fixed it across all experiments (i.e., attention with weak diagonal for matrix K, 300 epochs with early termination if validation error increases).

We report two GRIMP configurations, GRIMP-FT and GRIMP-E, which use fastText and EmbDI embeddings as pre-trained features, respectively. The number of GNN layers $\mathcal{L}_{GNN}$, shared merge layers $\mathcal{L}_{Shared}$, task-specific linear layers $\mathcal{L}_{Lin}$ is 2; the number of parameters in each GNN layer $\#P_{GNN}$ is 64, the number of parameters in linear layers $\#P_{Lin}$ is 128. The size of matrices Q and K is $(|C|)^3$ and $(|C|)^2$ respectively. Weight matrices W_Q and W_K have $\#P_W = \#P_{Lin} \cdot |C|$ parameters. Finally, the number of shared parameters is $\#P_s = \mathcal{L}_{GNN} \cdot |C| \cdot \#P_{GNN} + \mathcal{L}_{Shared} \cdot \#P_{Lin}$; the total number (including shared parameters) of parameters in linear models is $\Sigma P_l = \#P_s + |C| \cdot \#P_{Lin} \cdot \mathcal{L}_{Lin}$, while the total number of parameters of attention-based models is $\Sigma P_a = \#P_s + (|C|)^3 + (|C|)^2 + 2 \cdot \#P_W$. Values for $\#P_S$, ΣP_l, and ΣP_a are reported in Table 1.

Experiments have been conducted on a laptop with a CPU Intel i7-8550U, 8x1.8GHz cores and 32GB RAM.

4.2 Data Imputation Results

We test the imputation algorithms on ten clean datasets, which we corrupt by injecting increasing amounts of errors (5%, 20%, 50%). While our method is not designed assuming a specific missing data distributions, we inject missing values completely at random (MCAR) over the entire table, as it is the best practice to evaluate imputation [5, 45, 46, 52, 55]. The same dirty datasets are presented to every algorithm. The imputation accuracy is measured by comparing the dataset version imputed by each algorithm with the ground truth. Every injected missing value is used as test data. Edges for these test nodes are removed from the graph before training.

Comparison with Baselines. We report the imputation accuracy and training time in Figures 8 and 9, respectively. From the results in Figure 8, GRIMP is always among the top 3 methods and has an average rank of 1.6 (full ranking details in the extended version [12] Overall, EMBDI-MC is the worst performing algorithm. This is expected, as it uses EmbDI embeddings and a multiclass classifier, i.e., no multi task learning. EmbDI embeddings for cell values do not capture the complex data relationships that are

Dataset	Abbr.	# rows	# columns	$\lvert C\rvert$	$\lvert \mathcal{N}\rvert$	Distinct	#FD	S_{avg}	K_{avg}	F^+_{avg}	N^+_{avg}	$\#P_s$	ΣP_l	ΣP_a
Adult	AD	3016	14	9	5	289	2	2.6	13.3	0.7	2.9	2048	5632	8572
Australian	AU	690	15	9	6	957	0	2.7	24.0	0.6	7.5	2176	6016	9616
Contraceptive	CO	1473	10	8	2	65	0	0.0	-1.3	0.5	1.4	1536	4096	5196
Credit	CR	653	16	10	6	918	0	2.5	20.9	0.6	7.0	2304	6400	10752
Flare	FL	1066	13	10	3	34	0	0.4	-1.1	0.7	0.9	1920	5248	7614
IMDB	IM	4529	11	9	2	9829	0	7.2	220.2	0.5	83.2	1664	4480	5932
Mammogram	MM	830	6	5	1	93	0	0.6	-1.2	0.4	1.8	1024	2560	2812
Tax	TA	5000	12	5	7	910	6	2.1	12.1	0.5	7.5	1792	4864	6736
Thoracic	TH	470	17	14	3	255	0	0.3	-1.3	0.7	2.5	2432	6784	11986
Tic-Tac-Toe	TT	958	9	9	0	5	0	-0.2	-1.6	0.4	1.0	1408	3712	4522

Table 1: Statistics for all datasets. Abbreviation (Abbr.) is used instead of the full dataset name. $|C|$ and $|\mathcal{N}|$ state the number of categorical and numerical columns, respectively. Distinct is the number of unique values in the entire dataset. # FD is the number of FDs in a dataset. S_{avg} and K_{avg} are the coefficients of skewness and Kurtosis over the value distributions averaged over all columns, respectively. F^+_{avg} denotes the average fraction of rows that contain frequent values and N^+_{avg} is the average of the number of unique frequent values in every column. $\#P_s$ is the total number of parameters in the shared layer, ΣP_l and ΣP_a are the total number of task-specific parameters with linear tasks and attention tasks, respectively.

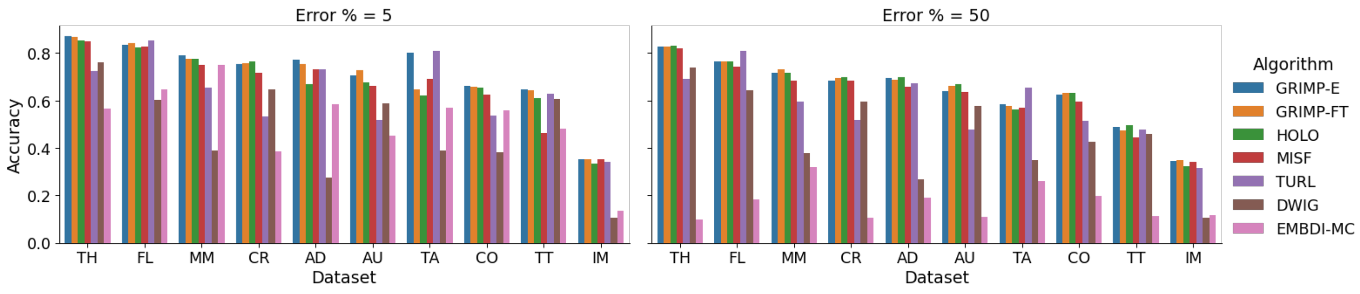


Figure 8: Imputation accuracy achieved by different baselines for all datasets.

needed for the imputation task. Interestingly, MissForest is competitive w.r.t. the best solutions in most cases. For numerical attributes and RMSE, Holoclean is the best performing method, GRIMP is comparable to MissForest, while TURL and Datawig report the worst results.

We also compute the *overall average imputation accuracy* by measuring the average imputation over all datasets for an algorithm. With 5% missing values, GRIMP with EmbDI obtains an average accuracy of 0.684, at least an 2% absolute increase over the state of the art methods Holoclean (0.665), TURL (0.608), and MissForest (0.648). Interestingly, even GRIMP linear, not reported in the plots, outperforms the baselines with an average accuracy of 0.676. All variants of GRIMP perform better than the state of the art solutions.

While both GRIMP and DataWig use the concept of embeddings, GRIMP routinely outperforms DataWig. This is due to three key factors. First, DataWig learns the embeddings of attributes in an independent manner: the embedding of the one attribute does not affect that of another. In contrast, GRIMP represents the entire relation as a graph and the embedding of every cell value is influenced by other cell values from the same tuple/attribute and those in the graph neighborhood. Second, the embedding learner of DataWig is relatively simple as it uses LSTM or n-gram hashing for strings. The learned embeddings are also not task specific, this applies also for EMBDI-MC. In contrast, our embeddings are learned in an imputation specific manner and are influenced by the embeddings of similar cell values through the process of message processing. Finally, DataWig does not use multi task learning, as it trains different classifiers for imputing each output attribute. Furthermore, each classifier uses a single loss as it only seeks to impute a single attribute. In contrast, GRIMP uses multi-task paradigm where multiple classifiers are learned using a two step process involving the learning of both *shared* and *task specific* embeddings.

While TURL is competitive in some cases, GRIMP leads to better accuracy in others and is better on average. TURL does worse for numerical attributes, as those are not considered in the original design. Indeed, TURL has been proposed for entity-focused tables and pretrained to learn factual knowledge using Wikipedia tables.

From the point of view of the execution time (reported in Figure 9), GRIMP with attention is often (but not always) the slowest algorithm, although Datawig is sometimes slower. MissForest is always among the fastest systems. TURL, which has been executed by its authors, took between 10 and 20 minutes to train for any dataset. GRIMP with linear tasks is comparable in execution time to the faster algorithms. The execution time of GRIMP and Holoclean decreases as the fraction of missing values increases. With a larger fraction of missing values, fewer viable cells remain, thus both algorithms work over a smaller quantity of data and terminate earlier. This is mirrored by the other baselines, MissForest and Datawig, whose models train longer in high-error configurations.

GRIMP Ablation. Experimental results for GRIMP when disabling different modules are reported in Figure 10. As expected the proposed modules have a significant impact on the accuracy.

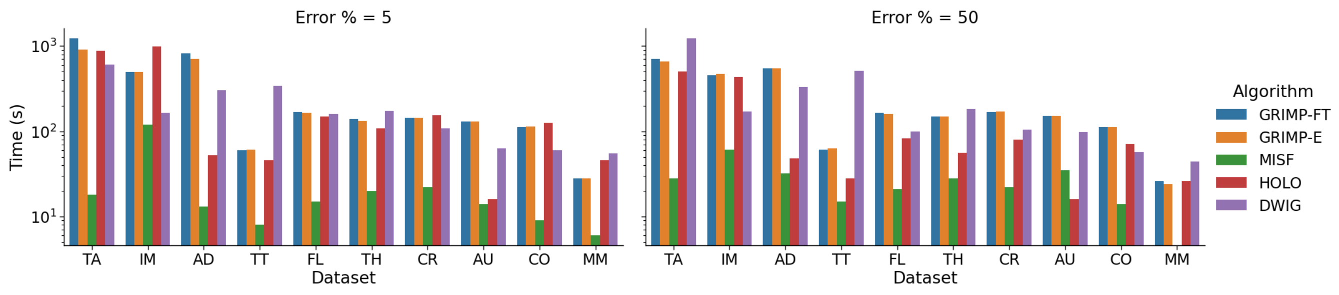


Figure 9: Training time in seconds required by the different baselines.

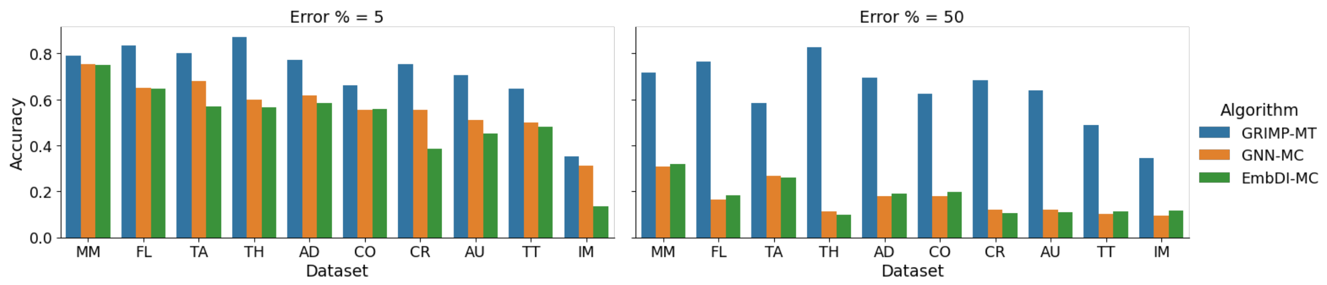


Figure 10: Results for GRIMP **with all modules enabled (GRIMP-MT), with GNN enabled and multi task learning disabled (GNN-MC), and with both GNN and multi task learning disabled (EmbDI-MC).**

Error %	Strategy	Accuracy	Time
5	Attention	0.707	307
	Linear	0.700	26
20	Attention	0.679	294
	Linear	0.671	28
50	Attention	0.637	258
	Linear	0.618	27

Table 2: Comparison of attention and linear tasks.

We also analyze the results obtained by GRIMP comparing attention and linear tasks. In the default "Attention" case, it uses the attention structures to combine attribute embeddings with the tuple embeddings from the GNN. In the "Linear" cases, each task consists only of fully connected linear layers. Results over all datasets in Table 2 show that Attention leads to higher accuracy. On the flip side, Linear is faster. Finally, while executions based on EmbDI features perform best on average, neither of the two pre-trained features clearly surpass the other in all settings. Both solutions slightly outperform the random initialization.

Impact of Noise in the Dataset. We also measure how robust is data imputation to noise in the original dataset. In this experiment, we first modify the values in the cells to have 10% of them as typos, i.e., every cell value has 10% probability to change its value by inserting random characters. We then run our experiment by removing 5% of the values, run imputation, and compute the final accuracy. Thanks to the inductive nature of GRIMP, its results show limited impact with an absolute decrease in accuracy of 0.062% with 10% of typos injected in the dataset.

4.3 Imputation with Input FDs

We evaluate our variant that consumes FDs as input, with two more baselines. First, we use the FDs to impute the values according to the minimality principle in data repairing [15]. For a null value involved in the conclusion (right-hand side) of a FD, FD-Repair imputes it with the most common value across the tuples with the same values in the premise (left-hand side) of the FD. For the second baseline, we extended MissForest to also use FDs. MissForest trains a random forest classifier with decision trees covering multiple random combinations of attributes. Because of the random selection, a fraction of the budget may be allocated to spurious combinations. We develop a FunForest variant by "pointing" the decision trees at the subset of attributes involved in FDs, thus reducing the noise introduced by unrelated columns. We allocate part of the budget to decision trees that are trained exclusively on FD attributes, while the rest of the trees are handled as in the original algorithm. For multiple FDs, the budget is split equally among them. Empirically, 50% of the budget for FDs leads to the best performance.

We focus on two datasets with FDs: Adult and Tax, with two (over two attributes) and six FDs (over ten attributes), respectively. We increase the fraction of injected errors from 5 to 50% over all attributes, measuring training time and imputation accuracy.

Results in Table 3 show that FD-Repair has the worst performance. This is expected, as the FDs cover only a subset of attributes, leading to imputation with high precision, but poor recall. FunForest is an improvement over the original MissForest when FDs are available. It improves by up to an absolute 10% the accuracy, whilst at the same time speeding up the convergence time of the algorithm by at least 50%. Training GRIMP-A with attention tasks on the FDs' attributes also improves the imputation accuracy w.r.t. the original setting. GRIMP-A outperforms FD-Repair, MissForest, and FunForest on the Adult dataset, while the random forest methods work better on the Tax dataset with

		Training time			Imputation accuracy			
Data	Error	MISF	FUNF	GRI-A	FD	MISF	FUNF	GRI-A
AD	5%	13.03	**2.38**	496.60	0.160	0.733	0.737	**0.766**
AD	20%	25.70	**6.05**	551.22	0.115	0.727	0.732	**0.756**
AD	50%	22.50	**15.23**	537.90	0.074	0.657	0.674	**0.693**
TA	5%	17.47	**6.00**	1117.54	0.386	0.689	0.786	**0.808**
TA	20%	23.18	**7.62**	977.62	0.309	0.661	**0.757**	0.632
TA	50%	27.94	**16.44**	751.93	0.194	0.571	**0.630**	0.586

Table 3: Accuracy for FD-Repair (FD), MissForest (MISF), FunForest (FUNF), and GRIMP-A (GRI-A) with input FDs.

	S_{avg}	K_{avg}	F^+_{avg}	N^+_{avg}
ρ	-0.467	-0.655	0.536	-0.660

Table 4: Pearson Correlation Coefficient (ρ) between the metrics reported in Table 1 and the imputation accuracy achieved by GRIMP **over all datasets.**

higher error rates. The results confirm that GRIMP is indeed able to learn more relationships, such as FDs, compared to the baselines.

While we use FDs as external information, both GRIMP and FunForest can use any attribute-related external information.

5 HITTING THE LIMITS OF IMPUTATION?

Inspired by the results of our experimental campaign, we study the limits of imputation models. We observe that, while the imputation accuracy for all algorithms varies across the datasets, the results are comparable across different methods. To explain this observation, we introduce four metrics that capture the data properties for each dataset, as reported in Table 1. The metrics are based on value distribution. The frequency of every unique value is measured, then every metric is computed over the distribution of frequencies in each column. Finally, the values for a metric obtained for each column are averaged to have a single figure for each dataset.

We compute two standard metrics, Fisher-Pearson coefficient of skewness (S_{avg}) and Kurtosis (as defined by Fisher) (K_{avg}); we then introduce two additional metrics that hinge on the distribution of *frequent values* in the data: F^+_{avg} and N^+_{avg}. A value is considered *frequent* if its number of occurrences is larger than the 90% quantile of all occurrence frequencies in the same attribute. Metric F^+ denotes the fraction of rows in a column that contain frequent values, while N^+ describes the number of unique frequent values in a column. Intuitively, these metrics give an idea of "how hard" it is to impute values in a dataset. Frequent values are in general easier to impute correctly, therefore a large value of F^+_{avg} implies that most rows in the dataset contain "easier" values. In contrast, N^+ measures how many distinct "frequent" values are present on average; in this case, a larger value means that there are several values to be considered "frequent", whereas figures closer to 1 describe situations in which very unique values are considered "frequent".

Our goal is to identify if there is a measure, based only on the value distribution in the dataset, that can predict the performance of the imputation methods. For example, both Thoracic and Flare have a high F^+_{avg} and a low N^+_{avg}, suggesting that, on average, there are few frequent values that dominate the dataset; conversely, IMDB has a low F^+_{avg} and a high N^+_{avg} as it contains mostly unique values (movie titles, actor, and director names). The metrics agree with the general imputation performance achieved by all methods, as both Thoracic and Flare are among the datasets with the highest imputation scores, whereas IMDB is among those with the lowest.

We start by looking at possible correlation between the metrics and the quality results for GRIMP. Table 4 reports the Pearson Correlation Coefficient ρ measured between the different measures and GRIMP's imputation accuracy on all datasets in the case with 50 % of missing values. We report the higher rate of missing values as it maximises the effect of missing values and highlights correlations. We observe a noticeable correlation between some of the metrics (in particular, K_{avg} and N^+_{avg}) and the imputation accuracy of the system. As the correlation is negative, this means that higher values of K_{avg} and N^+_{avg} lead to lower imputation results. This agrees with our intuition: better results are obtained when the distribution of values in the dataset is skewed towards few, very frequent values.

Indeed, imputation algorithms are biased towards frequent values, and thus have higher imputation accuracy for them, while they underperform on the rare values. We model this hypothesis by defining the "expected fraction of incorrect imputations" of a value v in a column A as $E^A_v = 1 - f_v$, where f_v is the frequency of value v in column A. The expectation is that a generic imputation algorithm will fail most imputations on rare values, while performing better on "easier" (i.e., more frequent) values.

In Figures 11 and 12, we plot the results of our experiments on the Thoracic and Contraceptive datasets, respectively. We report attributes with a small active domain to show that the issues arise even with in a simple setting. These two examples are representative for the other attributes and datasets. Each subplot contains data relative to a single attribute in the table, with the fraction of incorrectly imputed values on the y-axis (a value of 0 on the y-axis denotes perfect imputation accuracy for that value, thus the lower the bars, the better), and the different values in the attribute domain on the x-axis, sorted in descending order by frequency so that rare values in the attribute domain are on the right side of the plot. The blue bar (labeled as "expected") displays the expected fraction of erroneous imputations given a value's frequency as defined above, while the other bars show the actual fraction of erroneous imputations as they are produced by the different imputation algorithms. Our hypothesis is confirmed by the results: all algorithms tend to have a very high accuracy on frequent values, while failing frequently on rarer values. While different algorithms may exhibit different behaviors over different columns, there is a clear trend that is common to all of them.

In this analysis, we focus on frequency and skewness and leave the study of techniques to detect correlation across attributes to future work. While these results are preliminary, it is interesting to observe how algorithms that employ unrelated technologies exhibit remarkably similar behaviors in the imputation performance. We plan to delve deeper in this subject, with the objective of reaching a better understanding of how these systems model imputation and finding a heuristic that would allow to provide advice on what algorithm to use given a dataset's characteristics.

6 RELATED WORK

There are a variety of missing data imputation methods [34], ranging from imputing with the most common categorical value (or global average for numerical variables) [26] to K-Nearest

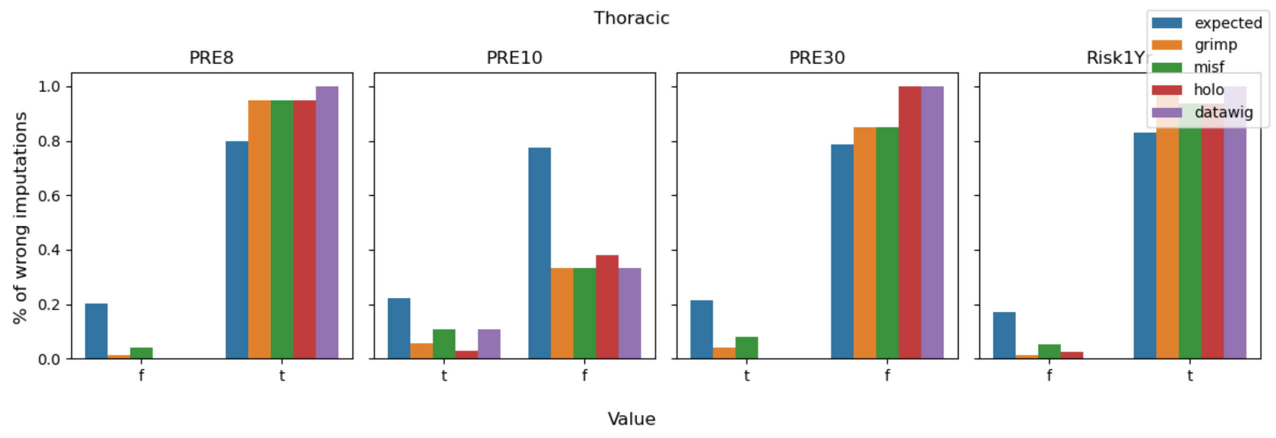


Figure 11: Distribution of wrong imputations in four attributes of the "Thoracic" dataset. All attributes have only two values ("f", "t") in the active domain. Frequent value (e.g., "f" in attribute PRE8) is correctly imputed by all methods, while rare value (e.g., "t" is PRE8) is problematic for all methods.

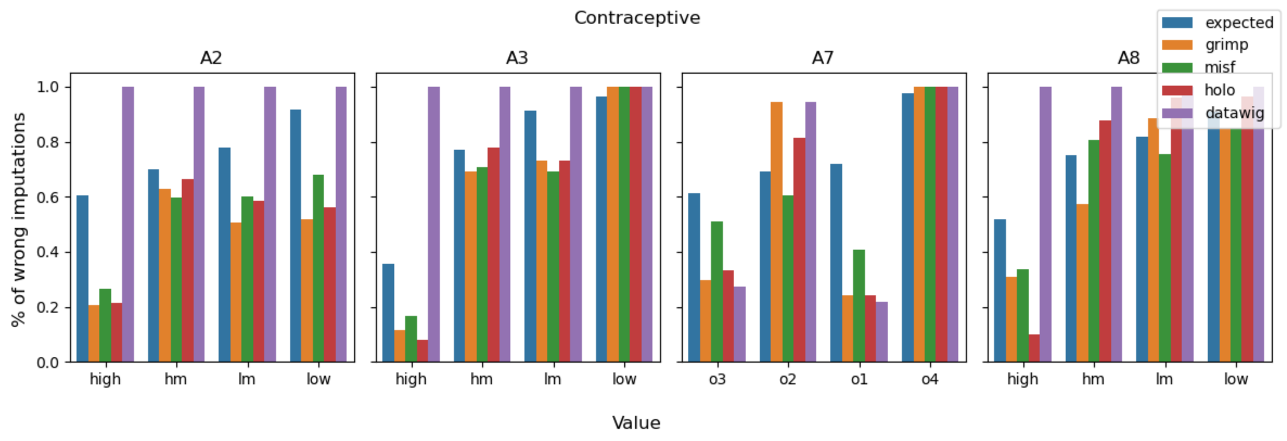


Figure 12: Distribution of wrong imputations in four attributes of the "Contraceptive" dataset. All attributes have four values in the active domain. Frequent values ("high", "o3") are better imputed than rarer ones by most methods.

Neighbor imputation [47] and *Rule-based methods* [16, 17]. Non-interpretable models include ML-based algorithms and can be distinguished in two categories, depending on how the imputation is generated. *Discriminative models* select a solution in the domain to impute missing values, e.g., MIDA [23], MICE [48], MissForest [46], and AimNet [52] (at the core of HoloClean). With *Generative models*, such as Generative Adversarial Networks or denoising autoencoders, the imputed version of the data is generated by the model. Generative models produce numerical outputs, so categorical values must be coerced to values in the active domain. Systems in this category include SVM-based imputation methods [30], HI-VAE [38], GAIN [54], and MIWAE [37]. GRIMP goes beyond the existing methods in several aspects. First, by handling mixed datasets, containing both categorical and continuous attributes, with its multi-task learning component [13, 57]. Second, GRIMP's self-supervised learning does not require a clean subset of the data for training. Third, its shared architecture exploits global relationships and statistics across tuples and dependencies across attributes. In our experiments, we also tested a table representation learning (TRL) system, TURL [19], for the missing value imputation task. In contrast with TURL and other TRL solutions based on pre-trained models [3, 29], our approach directly produces "metadata" embeddings for tuples and attributes [40].

Grape is a recent method formulating imputation as a link prediction problem [55]. Grape is close to GRIMP as they both use bipartite graphs for learning embeddings for the imputation downstream task. However, they have three main differences. First, Grape can impute only discrete or only continuous attributes; it cannot handle mixed data. Second, Grape formulates imputation as a link prediction task where the two end points are nodes corresponding to a tuple and feature, respectively. We can see of Grape's imputation as a regression/classification problem where the input features are the embeddings of the corresponding tuple and attribute. In contrast, the input to GRIMP is more expressive and contextual as it consists of the embeddings of *all* the cell values in that tuple. In other words, for a tuple with m attributes, GRIMP takes concatenated embeddings of $m-1$ cells while Grape takes only that of two. Finally, due to the link prediction based formulation, Grape does not provide any mechanism to learn dependencies between tuples and/or attributes. In contrast, GRIMP explicitly provides an attention based structure that learns data dependencies.

	GRIMP	EmbDI	DataWig	AimNet	Grape	TURL
Mixed data	Y	N	Y	Y	N	Partial
Graph rep. learn	Y	Y	N	N	Y	N
Attention	Y	N	N	Y	N	Y
Multi task learn	Y	N	N	Partial	N	Partial

Table 5: Comparison of GRIMP **and representative baselines.**

Relational data imputation is challenging [8, 24, 32, 42]. Relational data is heterogeneous with dense numerical features and sparse categorical features [28]; correlation between features is weaker than semantic or spatial relationships in text or image data [59]; variables can be correlated or independent; features have no positional information [44]. GRIMP tackles these challenges with a graph representation of the relational data and with a dual loss function that handles both categorical and numerical attributes as first class citizens. The graph enables the exploitation of Graph Neural Networks (GNNs), which offer an end-to-end solution for graph analysis tasks [53]. GRIMP uses a HeteroGNN model that combines different sub-modules, one for each table attribute. Each sub-module can use a different graph representation method that could be homogeneous, e.g., plain GCN [33], GraphSAGE [27], or heterogeneous [31, 41, 51]. Missing values can be harmful when data is used for training models [18, 21]. For this reason, data imputation is part of data cleaning tasks, where erroneous cells are first identified [1, 36, 43] and then imputed to repair the error [6, 35].

GRIMP is different from EmbDI [11] in a number of key dimensions. First, the GRIMP's graph reports modifications (such as self-loops) to achieve better results. Second, GRIMP does not use *tuple* representations for data imputation (i.e., embedding of tuple type nodes R_i) as in EmbDI. Instead, GRIMP relies on the embeddings of cell values. Third, EmbDI's embeddings are learned in an unsupervised manner in a task agnostic fashion. However, as we tested experimentally, the cell value embeddings learned by EmbDI are not sufficiently accurate for data imputation. In contrast, GRIMP learns embeddings for the specific downstream task of imputation with a GNN. Finally, EmbDI's the random walk based corpus generation approach preclude embeddings from incorporating data dependencies such as FDs. In contrast, GRIMP's attention structures explicitly *learn* complex data dependencies. We summarize the differences between GRIMP and existing methods in Table 5.

Finally, there has been some work to discern different ways to interpret missing values [39]. For example, using NULL for denoting information that is missing (i.e., unknown) versus for encoding that someone is not married (i.e., NULL for *Spouse* attribute). Existing work could be used as a pre-processing step for GRIMP to identify legitimate missing values.

7 CONCLUSIONS AND FUTURE WORK

We explored the problem of multiple imputation over relational data with mixed data types. GRIMP is a self-supervised imputation algorithm that combines a graph representation of the table based on GNNs and multi-task learning to produce state of the art results over a variety of datasets. We extended the tasks with an attention mechanism tailored to the multi-task architecture. Experiments demonstrate the promise of GRIMP. While our work shows good results, there remain a number of directions for future research.

On one side, we look at an improved solution. First, we plan to introduce hyperparameter tuning in the pipeline, so that GRIMP gets the optimal configuration for each dataset. Second, the ablation study shows that a GNN improves the accuracy w.r.t. simpler solutions, but its efficiency would benefit from optimizations such as graph pruning [2], reducing training data [60], and alternative message passing architectures [50]. Third, it would be interesting to extend the graph with external information, such as semantic annotations about the attributes [2, 56]. Fourth, as GRIMP is inductive, we plan to study how, once it is trained on one dataset, it can be reused on other datasets. Fifth, in settings where data privacy is an issue, we see GRIMP as a step that can lead to novel solutions for federated imputation [4, 58]. Finally, we tested missing values inserted at random (MCAR), but GRIMP's data-driven solution can handle systematic errors (MNAR) as demonstrated by previous work [5, 52] - we plan to evaluate this scenario in follow-up work.

On the other side, we are interested in a broader study of DL-based imputation models. DL is not a silver bullet and suffers from some drawbacks [24, 42]. We show that the imputation accuracy of various algorithms is approximated by the frequency of values in the data. While our results are preliminary, we observe how algorithms that employ unrelated technologies tend to exhibit similar behavior. We plan to delve deeper in this subject, as we believe this effort will enable a better understanding of how systems model imputation and what are the intrinsic limits to tackle in future research.

Acknowledgement. This work has been partially supported by the ANR grant ANR-18-CE23-0019 and by the IMT Futur & Ruptures program "AutoClean".

REFERENCES

[1] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. 2016. Detecting data errors: Where are we and what needs to be done? *Proceedings of the VLDB Endowment* 9, 12 (2016), 993–1004.

[2] Naser Ahmadi, Hansjorg Sand, and Paolo Papotti. 2022. Unsupervised Matching of Data and Text. In *2022 IEEE ICDE.*

[3] Gilbert Badaro, Mohammed Saeed, and Papotti Paolo. 2023. Transformers for Tabular Data Representation: A Survey of Models and Applications. *Transactions of the Association for Computational Linguistics* 11 (2023), 227–249. https://doi.org/doi.org/10.1162/tacl_a_00544

[4] Irene Balelli, Aude Sportisse, Francesco Cremonesi, Pierre-Alexandre Mattei, and Marco Lorenzi. 2023. Fed-MIWAE: Federated Imputation of Incomplete Data via Deep Generative Models. *arXiv preprint arXiv:2304.08054* (2023).

[5] Felix Biessmann, Tammo Rukat, Philipp Schmidt, Prathik Naidu, Sebastian Schelter, Andrey Taptunov, Dustin Lange, and David Salinas. 2019. DataWig: Missing Value Imputation for Tables. *J. Mach. Learn. Res.* 20 (2019), 175–1.

[6] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2007. Conditional functional dependencies for data cleaning. In *2007 IEEE 23rd international conference on data engineering*. IEEE, 746–755.

[7] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics* 5 (2017), 135–146.

[8] Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. 2021. Deep Neural Networks and Tabular Data: A Survey. *CoRR* abs/2110.01889 (2021). arXiv:2110.01889 https://arxiv.org/abs/2110.01889

[9] Ramiro D Camino, Christian A Hammerschmidt, and Radu State. 2019. Improving missing data imputation with deep generative models. *arXiv preprint arXiv:1902.10666* (2019).

[10] Riccardo Cappuzzo. 2022. GRIMP code and data. https://gitlab.eurecom.fr/cappuzzo/GNNforImputation.

[11] Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. 2020. Creating Embeddings of Heterogeneous Relational Datasets for Data Integration Tasks. In *SIGMOD* (Portland, OR, USA). ACM, 15 pages.

[12] Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. 2023. Relational Data Imputation with Graph Neural Networks. https://www.eurecom.fr/~papotti/files/GrimpTR.pdf.

[13] Rich Caruana. 1997. Multitask learning. *Machine learning* 28, 1 (1997), 41–75.

[14] Xu Chu, Ihab F Ilyas, and Paolo Papotti. 2013. Discovering denial constraints. *Proceedings of the VLDB Endowment* 6, 13 (2013), 1498–1509.

[15] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Holistic data cleaning: Putting violations into context. In *IEEE*. IEEE Computer Society, 458–469.
[16] Peter Clark and Tim Niblett. 1989. The CN2 induction algorithm. *Machine learning* 3, 4 (1989), 261–283.
[17] William W Cohen. 1995. Fast effective rule induction. In *Machine learning proceedings 1995*. Elsevier, 115–123.
[18] Antonio de la Vega de León, Beining Chen, and Valerie J Gillet. 2018. Effect of missing data on multitask prediction methods. *Journal of cheminformatics* 10, 1 (2018), 1–12.
[19] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. Turl: Table understanding through representation learning. *arXiv preprint arXiv:2006.14806* (2020).
[20] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml
[21] Alireza Farhangfar, Lukasz Kurgan, and Jennifer Dy. 2008. Impact of imputation of missing values on classification error for discrete data. *Pattern Recognition* 41, 12 (2008), 3692–3705.
[22] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *International conference on machine learning*. PMLR, 1263–1272.
[23] Lovedeep Gondara and Ke Wang. 2018. Mida: Multiple imputation using denoising autoencoders. In *Pacific-Asia conference on knowledge discovery and data mining*. Springer, 260–272.
[24] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. 2021. Revisiting Deep Learning Models for Tabular Data. *CoRR* abs/2106.11959 (2021). arXiv:2106.11959 https://arxiv.org/abs/2106.11959
[25] Palash Goyal and Emilio Ferrara. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems* 151 (2018), 78–94.
[26] Jerzy Grzymala-Busse, Linda Goodwin, Witold Grzymala-Busse, and Xinqun Zheng. 2005. Handling missing attribute values in preterm birth data sets. In *Int. Workshop on Rough Sets, Fuzzy Sets, Data Mining, and Granular-Soft Computing*.
[27] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 1025–1035.
[28] John T. Hancock and Taghi M. Khoshgoftaar. 2020. Survey on categorical data for neural networks. *J. Big Data* 7, 1 (2020), 28. https://doi.org/10.1186/s40537-020-00305-w
[29] Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. TaPas: Weakly supervised table parsing via pre-training. *arXiv preprint arXiv:2004.02349* (2020).
[30] Feng Honghai, Chen Guoshun, Yin Cheng, Yang Bingru, and Chen Yumei. 2005. A SVM regression based approach to filling in missing values. In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*. Springer, 581–587.
[31] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020*. 2704–2710.
[32] Arlind Kadra, Marius Lindauer, Frank Hutter, and Josif Grabocka. 2021. Regularization is all you Need: Simple Neural Nets can Excel on Tabular Data. *CoRR* abs/2106.11189 (2021). arXiv:2106.11189 https://arxiv.org/abs/2106.11189
[33] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[34] Julián Luengo, Salvador García, and Francisco Herrera. 2012. On the choice of the best imputation methods for missing values considering three groups of classification methods. *Knowledge and information systems* 32, 1 (2012), 77–108.
[35] Mohammad Mahdavi and Ziawasch Abedjan. 2020. Baran: Effective error correction via a unified context representation and transfer learning. *Proceedings of the VLDB Endowment* 13, 12 (2020), 1948–1961.
[36] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha: A configuration-free error detection system. In *Proceedings of the 2019 International Conference on Management of Data*. 865–882.
[37] Pierre-Alexandre Mattei and Jes Frellsen. 2019. MIWAE: Deep generative modelling and imputation of incomplete data sets. In *ICML*. PMLR, 4413–4423.
[38] Alfredo Nazabal, Pablo M Olmos, Zoubin Ghahramani, and Isabel Valera. 2020. Handling incomplete heterogeneous data using vaes. *Pattern Recognition* 107 (2020), 107501.
[39] Abdulhakim A. Qahtan, Ahmed Elmagarmid, Raul Castro Fernandez, Mourad Ouzzani, and Nan Tang. 2018. FAHES: A Robust Disguised Missing Values Detector. In *SIGKDD (KDD '18)*. Association for Computing Machinery, New York, NY, USA, 2100–2109. https://doi.org/10.1145/3219819.3220109
[40] Mohammed Saeed and Paolo Papotti. 2022. You Are My Type! Type Embeddings for Pre-trained Language Models. In *EMNLP*. Association for Computational Linguistics, 4583–4598. https://doi.org/10.18653/V1/2022.FINDINGS-EMNLP.336
[41] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European semantic web conference*. Springer, 593–607.
[42] Ravid Shwartz-Ziv and Amitai Armon. 2021. Tabular Data: Deep Learning is Not All You Need. *CoRR* abs/2106.03253 (2021). arXiv:2106.03253 https://arxiv.org/abs/2106.03253
[43] Stavros Sintos, Pankaj K. Agarwal, and Jun Yang. 2019. Selecting Data to Clean for Fact Checking: Minimizing Uncertainty vs. Maximizing Surprise. *Proc. VLDB Endow.* 12, 13 (2019), 2408–2421.
[44] Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C. Bayan Bruss, and Tom Goldstein. 2021. SAINT: Improved Neural Networks for Tabular Data via Row Attention and Contrastive Pre-Training. *CoRR* abs/2106.01342 (2021). arXiv:2106.01342 https://arxiv.org/abs/2106.01342
[45] Indro Spinelli, Simone Scardapane, and Aurelio Uncini. 2020. Missing data imputation with adversarially-trained graph convolutional networks. *Neural Networks* 129 (2020), 249–260.
[46] Daniel J Stekhoven and Peter Bühlmann. 2012. MissForest—non-parametric missing value imputation for mixed-type data. *Bioinformatics* 28, 1 (2012), 112–118.
[47] Olga Troyanskaya, Michael Cantor, Gavin Sherlock, Pat Brown, Trevor Hastie, Robert Tibshirani, David Botstein, and Russ B Altman. 2001. Missing value estimation methods for DNA microarrays. *Bioinformatics* 17, 6 (2001), 520–525.
[48] Stef Van Buuren and Karin Groothuis-Oudshoorn. 2011. mice: Multivariate imputation by chained equations in R. *Journal of statistical software* 45 (2011), 1–67.
[49] Simon Vandenhende, Stamatios Georgoulis, Marc Proesmans, Dengxin Dai, and Luc Van Gool. 2020. Revisiting multi-task learning in the deep learning era. *arXiv preprint arXiv:2004.13379* 2 (2020).
[50] Roger Waleffe, Jason Mohoney, Theodoros Rekatsinas, and Shivaram Venkataraman. 2023. MariusGNN: Resource-Efficient Out-of-Core Training of Graph Neural Networks. In *EuroSys*.
[51] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous graph attention network. In *The World Wide Web Conference*. 2022–2032.
[52] Richard Wu, Aoqian Zhang, Ihab Ilyas, and Theodoros Rekatsinas. 2020. Attention-based learning for missing data imputation in holoclean. *Proceedings of Machine Learning and Systems* 2 (2020), 307–325.
[53] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems* 32, 1 (2020), 4–24.
[54] Jinsung Yoon, James Jordon, and Mihaela Schaar. 2018. Gain: Missing data imputation using generative adversarial nets. In *ICML*. PMLR, 5689–5698.
[55] Jiaxuan You, Xiaobai Ma, Daisy Yi Ding, Mykel J. Kochenderfer, and Jure Leskovec. 2020. Handling Missing Data with Graph Representation Learning. In *NeurIPS 2020*.
[56] Dan Zhang, Yoshihiko Suhara, Jinfeng Li, Madelon Hulsebos, Çağatay Demiralp, and Wang-Chiew Tan. 2020. Sato: Contextual Semantic Type Detection in Tables. *Proc. VLDB Endow.* 13, 11 (2020), 1835–1848.
[57] Yu Zhang and Qiang Yang. 2021. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering* (2021).
[58] Xu Zhou, Xiaofeng Liu, Gongjin Lan, and Jian Wu. 2021. Federated conditional generative adversarial nets imputation method for air quality missing data. *Knowledge-Based Systems* 228 (2021), 107261.
[59] Yitan Zhu, Thomas Brettin, Fangfang Xia, Alexander Partin, Maulik Shukla, Hyunseung Yoo, Yvonne A Evrard, James H Doroshow, and Rick L Stevens. 2021. Converting tabular data into images for deep learning with convolutional neural networks. *Scientific reports* 11, 1 (2021), 1–11.
[60] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. 2019. Layer-dependent importance sampling for training deep and large graph convolutional networks. *Advances in neural information processing systems* 32 (2019).