Permutation Invariant Strategy Using Transformer Encoders for Table Understanding

Sarthak Dash, Sugato Bagchi,
Nandana Mihindukulasooriya, Alfio Gliozzo
IBM Research AI, Thomas J. Watson Research Center
Yorktown Heights, NY

Abstract

Representing text in tables is essential for many business intelligence tasks such as semantic retrieval, data exploration and visualization, and question answering. Existing methods that leverage pretrained Transformer encoders range from a simple construction of pseudo-sentences by concatenating text across rows or columns to complex parameter-intensive models that encode table structure and require additional pretraining. In this work, we introduce a novel encoding strategy for Transformer encoders that preserves the critical property of permutation invariance across rows or columns. Unlike existing state-of-the-art methods for Table Understanding, our proposed approach does not require any additional pretraining and still substantially outperforms existing methods in almost all instances. We demonstrate the effectiveness of our proposed approach on three table interpretation tasks: column type annotation, relation extraction, and entity linking through extensive experiments on existing tabular datasets.

1 Introduction

Representation learning of natural language text within relational databases, spreadsheets, and other structured content has received particular interest in recent times due to the advances made in Transformer-based language models (Devlin et al., 2019; Lan et al., 2020; Raffel et al., 2020). There is a growing need for developing automated systems to generate insights and make decisions with information in these sources. Along with quantitative data, these sources also have textual content that play a crucial role in tasks such as retrieval from data catalogs (Zhang and Balog, 2018), question answering (Glass et al., 2021; Yin et al., 2020; Herzig et al., 2020), and automating business intelligence tasks (Sallam and Idoine, 2019).

The availability of large pretrained language models using Transformers has enabled researchers

Name	**Party**	**Riding**
Brad Cathers	Yukon	Lake Laberge
Nils Clarke	Liberal	Riverdale North
Yvonne Clarke	Yukon	Porter Creek Centre
Sandy Silver	Liberal	Klondike
Jeremy Harper	Liberal	Mayo-Tatchun

Table 1: An example partial table from Wikipedia having *pageTitle* YUKON LEGISLATIVE ASSEMBLY, and *sectionTile* CURRENT MEMBERS.

to encode text within relational tables in many ways. In one approach, existing works (Glass et al., 2021; Yin et al., 2020) on Question Answering over Relational Tables build a pseudo-sentence by concatenating row/column entries, for example,

> **[S1]** Name: Brad Cathers | Nils Clarke | ... | Jeremy Harper
> **[S2]** Name : Brad Cathers | Party : Yukon | Riding: Lake Laberge

which corresponds to the first column and the first row respectively in Table 1. Here, the tokens | and : act as delimiters. Such pseudo-sentences are then processed through a Transformer encoder model, and the `[CLS]` vector at the final layer is treated as its representation (Devlin et al., 2019). While such a strategy provides valuable context-based information, it is sensitive to the ordering of the cell values used in the encoding process.

Consider the pseudo-sentence **[S3]** which is built by randomly shuffling the values in **[S1]**,

> **[S3]** Name : Nils Clarke | ... | Jeremy Harper | Brad Cathers

If we follow above strategy, the final representation for **[S1]** and **[S3]** will be different. This would be sub optimal, since it would be at odds with the tabular structure where rows and columns may be shuffled without losing semantic meaning.

Another approach used to encode text within a relational table is introduced by Deng et al. (2020).

Here, the authors employ table metadata and the row-column structure to constrain self-attention only over structurally related elements. For example, the mention *Brad Cathers* in Table 1 can only attend to entries within the same row/column and to the table metadata such as headers and captions. Such constraints introduce additional table pretraining objectives for representation learning, which comes with a relatively high up-front computational cost. These constraints and representation are also inflexible to the explicit consideration of structural relationships across rows/columns, such as relations between column-pairs or hierarchies between rows/columns that may need to be leveraged in some end tasks.

In this paper, we propose a *permutation invariant* position encoding strategy that we call PI Strategy, which can be used with existing Transformer-based language models. The PI Strategy uses the pseudo-sentence construction approach while allowing the model to be insensitive to the ordering of cells within a row or column. We evaluate our approach on three relational table interpretation tasks: column type annotation, column relation linking, and cell entity linking. Compared to existing state-of-the-art approaches, we show that:

- Our approach is less parameter intensive.
- Our approach can also adapt the pseudo-sentence representation to the requirements of the downstream tasks.
- Our approach without any pretraining outperforms existing state-of-the-art approaches on almost all instances.

2 Related Work

The task of modeling *set-input* problems, i.e., data instances that behave as a set rather than a sequence, using neural networks has been slowly gaining traction. Recent works such as Edwards and Storkey (2017); Zaheer et al. (2017) propose a *three* step strategy. First, each set element is encoded independently to a fixed-size embedding. The second step comprises of a commutative *pooling* operation, and finally, the pooled embedding is processed through a non-linear layer. Because each set element is processed independently, information regarding possible interactions between the set elements has to be necessarily discarded (Lee et al., 2019).

Vinyals et al. (2016) handle set inputs by *pooling* them via a weighted average operation with weights computed via an attention mechanism. Yang et al. (2020) employ a similar strategy for multi-view 3D reconstruction, where a dot-product attention operator is used for weighted average *pooling*. Ilse et al. (2018) use attention-based weighted sum-pooling for multiple-instance learning. Santoro et al. (2017) propose the relational network, an architecture that sum-pools all pairwise interactions between set elements, but not the higher order interactions. Compared to these approaches, ours uses multi-head attention for aggregation. Also, our approach uses stacks of self-attention modules within the Transformer encoder, enabling us to model higher-order interactions between textual instances in a table row/column.

Lee et al. (2019) introduce Set transformer, a novel Transformer architecture, wherein the encoder module does not contain positional embedding and drop-out features. Such an approach is not apt for encoding rows/columns within a table because the ordering of individual tokens within a particular cell value must still be encoded.

In the area of the representation learning strategies on tables, recent works such as TAPAS (Herzig et al., 2020), and TaBERT (Yin et al., 2020) encode the natural language question, query table pair jointly, and employ a transformer encoder model to assist in semantic parsing or question answering (QA) over tables. TAPAS proposes multiple additional embeddings to encode the entire tabular structure. However, none of these embeddings is designed to encode permutation invariance. For example, the position embeddings in TAPAS are monotonically increasing from left to right, thereby encoding a sense of ordering across the cell mentions. Moreover, the Row ID and the Column ID embeddings within TAPAS impart a sense of ordering across the rows and columns. Another table encoding model MATE (Eisenschlos et al., 2021) is still vulnerable to row and column perturbations.

On the other hand, understanding relational tables by mapping them to entities, types, and relations in a semantically rich knowledge base (KB) is a well-studied problem (Zhang and Balog, 2020; Ritze et al., 2015). The main table interpretation tasks involve table cell linking, column type annotation, and relation extraction between column pairs. Recent works such as TCN (Wang et al., 2021) and DODUO (Suhara et al., 2021) employ multi-task training objective for jointly learning column types and relation labels. TABBIE (Iida et al., 2021) uses

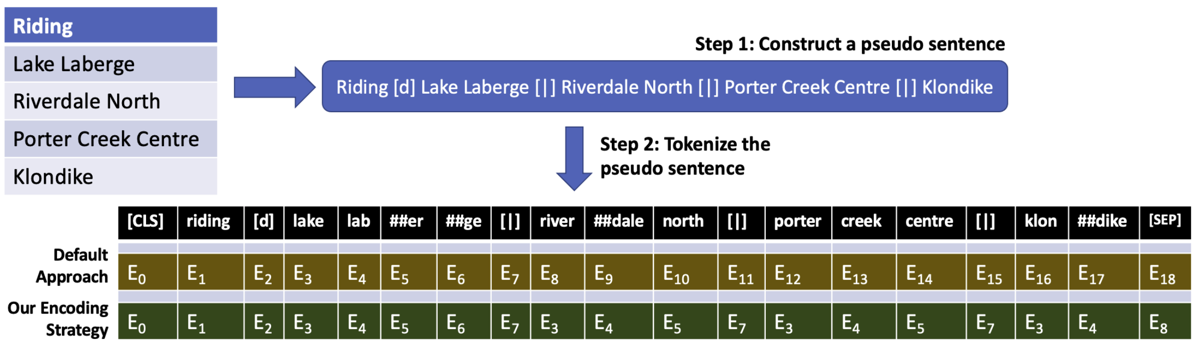
Riding
Lake Laberge
Riverdale North
Porter Creek Centre
Klondike
Step 1: Construct a pseudo sentence
Riding [d] Lake Laberge [|] Riverdale North [|] Porter Creek Centre [|] Klondike
Step 2: Tokenize the pseudo sentence
[CLS] riding [d] lake lab ##er ##ge [|] river ##dale north [|] porter creek centre [|] klon ##dike [SEP]
Default Approach
E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 E10 E11 E12 E13 E14 E15 E16 E17 E18
Our Encoding Strategy
E0 E1 E2 E3 E4 E5 E6 E7 E3 E4 E5 E7 E3 E4 E5 E7 E3 E4 E8

Figure 1: A graphical representation of our Permutation Invariant position encoding strategy. Given a *query* column with a header we first construct a pseudo sentence, and then tokenize it. While standard Transformer-based language models such as BERT, ALBERT, GPT2, etc. assign *position ids* from left-to-right (Default Approach), our approach assigns them in a piece-wise monotonically increasing manner.

two transformers, one to encode rows and the other to encode columns, and employs a corrupt cell detection objective function for pretraining. TABBIE was pretrained using eight v100 GPUs for a week.

TURL (Deng et al., 2020) introduces a *structure-aware* Transformer that employs a pretraining/fine-tuning strategy for the table understanding tasks. In contrast, our approach does not require additional pretraining. Therefore it saves upon the computational overhead associated with pretraining. Furthermore, we also show that our method performs better compared to TURL on the table interpretation tasks. Currently, TURL is the published state-of-the-art on the three table interpretation tasks and TCN (Wang et al., 2021) improves on TURL for a subset of tasks with data on both column type annotation and relation extraction.

3 Permutation Invariance

In this section, we first describe our *permutation-invariant* position encoding strategy, and then explain how it ensures permutation invariance. Figure 1 illustrates a graphical representation of our encoding strategy for a given *query* column. Current approaches (Devlin et al., 2019; Lan et al., 2020; Brown et al., 2020) use absolute position IDs from left-to-right as indicated by the “Default Approach". In contrast, we propose an alternative strategy for assigning position IDs that work as follows,

- We introduce two special tokens `[d]` and `[|]`. As shown in Figure 1, the `[|]` token acts as a delimiter between the cell values in a column, whereas the `[d]` separates the column header from the column values. Both `[d]` and `[|]` are initialized at random, and their representations are learned during training.
- Starting from a position ID of *zero* for the `[CLS]` token, we increment by *one* till we reach the `[d]` token. This assigns position IDs *zero* through *two* for this example.
- If the position ID assigned to the `[d]` token is n, then for all tokenized cell values, we start assigning position IDs from $n + 1$ onwards, in a left-to-right manner.
- In this example, the cell values *lake laberge* (four tokens) gets position IDs of *three* to *six*, whereas *Riverdale North* and *Porter Creek Centre* (three tokens each) gets position IDs of *three* to *five*. The cell value *Klondike* with two tokens gets position ID of *three* and *four*.
- If m is the max value of position ID assigned after all cells are processed, then all the `[|]` tokens get a position ID of $m + 1$, i.e., *seven* for this example.
- Finally, the `[SEP]` token gets a position ID of $m + 2$, i.e. a value of *eight* in this example.

Once we build the position IDs for a given pseudo-sentence ψ according to our strategy, we use it to encode ψ via a Transformer-based language model. A Transformer based language model typically consists of the encoder module of a transformer, which generally takes in the following as inputs, *a*) Sub-word embeddings for each sub–word within the tokenized sentence, *b*) Token type embeddings for each subword token, and *c*) Position Embeddings for each subword token. The position embeddings are responsible for assigning an ordering to the sequence of sub-word tokens since a transformer encoder module is unaware of a token's position. This ordering influences the context of the subword token. These three embeddings are added for each sub-word token in ψ, the transformer encoder then processes the resulting embedding sequence. Following the strategy used by Devlin et al. (2019) to fine-tune on GLUE benchmark, we consider the `[CLS]` vector as the aggregated representation of the *query* column.

For a given column c, let ψ_1 and ψ_2 denote two pseudo sentences due to two independent orderings of cell values. Under our encoding strategy, we observe that the vector corresponding to any token t (in either ψ_1 or ψ_2) attends[1] over the same set of other token vectors regardless of cell ordering. At each layer of the transformer encoder, both ψ_1 and ψ_2 generate the same set of intermediate vectors, just with a different ordering. Moreover, the first vector in either ordering corresponds to the `[CLS]` token and is equal for both ψ_1 and ψ_2. Therefore, the `[CLS]` vector at the final layer, which we refer to as our column representation, remains the same for ψ_1 and ψ_2, thereby yielding a permutation invariant encoding. Our encoding strategy ensures that all the cell values are positionally equidistant to the `[CLS]` token and expected to have comparable impact on its encoding.

Consider the example column in Figure 1. This column contains four cell values yielding twenty-four permutations in total. For each permutation, we computed its column representation as described above. Finally, we calculated the variance of all possible l2 distances between any two column representations. The variance measure for our proposed permutation invariant position encoding strategy was 1.5e-12, whereas, for the default approach, this value was 0.41. This indicates that our proposed approach yields embeddings that are invariant to the ordering of the cell-values.

4 Tasks

We first define the table interpretation tasks and then describe our approach using the PI Strategy.

4.1 Column Type Annotation

The column type annotation task is defined as,

Definition 1. *Given a table T and a set of semantic types $\mathcal{L}$, the column type annotation task refers to the task of annotating a column $c \in T$ with a type label $l \in \mathcal{L}$ so that all entities in c have the type l. Note that a column can have multiple type labels.*

Column type annotation is a crucial task for Table understanding and can provide vital information in many downstream tasks, such as Question Answering over Tables, Knowledge Discovery, etc. Earlier works (Mulwad et al., 2010; Ritze et al., 2015; Zhang, 2017) often coupled this task together with entity linking. The entities within a column were first linked to a KB, and then a majority voting strategy was employed on the types of the linked entities. Recently, Chen et al. (2019a,b); Hulsebos et al. (2019); Deng et al. (2020) have studied this task based on only the available information in a given table without doing entity linking first. In this work, we adopt a similar setting and compare against Deng et al. (2020), which is the current state of the art on this task.

Our Approach. Given a column c, for example, let's say the *second* column in Table 1, we encode it via the PI Strategy to obtain column vector $\mathbf{h}$. The probability of predicting the class label l is,

$$P(l) = \text{Sigmoid}(\mathbf{h}W_l + b_l) \qquad (1)$$

We use a binary cross-entropy loss function for training.

4.2 Relation Extraction

The relation extraction task is defined as follows,

Definition 2. *Given a table $\mathcal{T}$ and a set of relations $\mathcal{R}$ in a KB, for a subject-object column pair in $\mathcal{T}$, the goal is to annotate it with $r \in \mathcal{R}$, so that r holds between all the entity pairs.*

Similar to the Column Type Annotation task, existing approaches Mulwad et al. (2010); Ritze et al. (2015); Zhang (2017) perform entity linking first, followed by a KB lookup to obtain the list of applicable relations. These approaches rely on the entity linking performance and assume that the KB in question is complete, i.e., all relations between entity pairs are present in the KB. Such an assumption *may not* be true in general. Therefore, following Deng et al. (2020), the goal is to classify a given subject-object column pair into one or more relations, without explicitly linking table cell mentions to entities. This is *important* as it allows an user to extract new knowledge from web tables for *knowledge base population* tasks.

[1]We use a scaled dot product attention operator, which given a key-value pair (k, V) is invariant to the ordering of vectors in V.

Our Approach. To predict the relation label(s) for a given column pair p, q (let's say columns *one* and *two* in Table 1), we first concatenate the two columns to obtain a pseudo-column z, i.e. the i-th cell value of column z is the concatentation (separated by a special delimiter token `[:]`) of the i-th cell values of columns p and q.

We employ two different strategies to build a model for this task, based on whether we are using *entity mentions* only or using *extra metadata* too (See Table 2 below). For the case of *entity mentions* only, we encode the column z based on our PI Strategy to obtain an unique representation $\mathbf{h}$ for the column pair. Additionally, if we are using extra metadata too, then we encode columns p, q and z separately using the PI Strategy to obtain representations $\mathbf{h_p}, \mathbf{h_q}, \mathbf{h_z}$, and concatenate them to obtain the final representation $\mathbf{h}$.

The goal of the concatenation step is to ensure that as long as the cells in columns p, q are shuffled in tandem, the overall column pair representation remains the same. Finally, for predicting the class labels, we follow the same expression as in equation 1, and also employ binary cross-entropy loss function for training.

4.3 Zero shot Column Type Annotation

To further test the *effectiveness* of PI Strategy in generating meaningful column representations, we construct a Zero-shot column type annotation task.

To predict the type(s) for a given column in the zero-shot setting, we employ a *Siamese network* architecture, by using *two* separate transformer encoders. Each training instance consists of a (table column, list of type labels pair), i.e., $\mathcal{I} = (c, \mathcal{Y}_c)$ where c is a particular column, and $\mathcal{Y}_c$ is a list of type labels associated with c. We encode column c using the PI Strategy and the *first* transformer encoder. Concurrently, we encode the true type label $y \in \mathcal{Y}_c$ using the PI Strategy and the *second* transformer encoder to obtain the label representation $\mathbf{h_y}$. Finally, we calculate the positive score s^+ as the dot product between $\mathbf{h_c}$ and $\mathbf{h_y}$.

The negative type labels are sampled at *random*, and a similar process is used to calculate the negative score s^-. We use both s^+ and s^-, together with a binary cross-entropy loss for training.

4.4 Entity Linking

The entity linking task is defined as follows,

Definition 3. *Given a table T and a knowledge base $\mathcal{K}$, entity linking aims to assign each potential mention of cells in T to its referent entity $e \in \mathcal{K}$.*

Entity linking is usually addressed in a two-step approach: *a*) Candidate generation, and *b*) Entity disambiguation. The Candidate generation module proposes a set of potential candidates, whereas the Entity disambiguation module is a re-ranking step that selects the best candidate entity matching the cell mention for a given table.

Recent methods for table entity linking include T2K (Ritze et al., 2015) that uses an iterative matching approach, combining schema and entity matching; Hybrid II (Efthymiou et al., 2017) that combines a lookup and entity embedding method. Following Deng et al. (2020), we use the same Wikidata lookup service for candidate generation and focus only on the disambiguation step.

Our approach. Entity disambiguation is a matching problem that requires us to match a given table cell mention to a particular entity within a candidate set of entities. We employ a bi-encoder architecture to rank the candidate set of entities for a query mention. Given a cell mention m within a table T_m, we extract the corresponding row $\mathcal{R}_m$, which consists of a set of column header, cell-value pairs representing different attributes of m.

Consider the mention *Lake Laberge* in the first row of Table 1 as an example. We extract out this row and build the following pseudo-sentence:

> Yukon Legislative Assembly Current Members [m] [s] Riding [:] Lake Laberge [e] [d] Name [:] Brad Cathers [l] Party [:] Yukon

For this task, we introduce another *special* delimiter token `[m]` that separates the metadata information[2] from the rest of the pseudo sentence. In this case, the *page title*, *section title*, and the *table caption* (See Table 1) constitute the metadata information. Moreover, following Soares et al. (2019), we introduce two additional special tokens `[s]` and `[e]` that surround the query mention. The rest of pseudo-sentence after `[d]` consists of the remaining entries within $\mathcal{R}_m$.

This pseudo-sentence is then encoded according to our proposed PI Strategy where entries after `[d]` are permutationally invariant. This generates the representation $\mathbf{h_m}$, which is the output vector at the final layer, corresponding to the `[s]` token.

On the other hand, following Deng et al. (2020), for each candidate entity $e \in \mathcal{K}$, we use its name

[2]We concatenate the metadata together, which forms the beginning of our pseudo sentence.

N, description D, and a set of type labels T. We concatenate N, D together with the labels T, to obtain the pseudo-sentence, which is then encoded using PI Strategy to obtain a representation $\mathbf{h_e}$ for the candidate entity e.

Given the mention vector $\mathbf{h_m}$ and the candidate entity vector $\mathbf{h_e}$, we employ a similar training strategy as the zero-shot column type annotation task.

5 Evaluation

To be comparable to TURL, we implement our PI Strategy using a TinyBERT (Jiao et al., 2020) model. For comparing our approach against existing state-of-the-art methods, we use the datasets released by Deng et al. (2020). This paper introduces three datasets, one each for Column Type Annotation (CT), Relation Extraction (RE), and Entity Linking (EL) tasks. We refer to them as WT-TURL-{CT,RE,EL} respectively in *this* work.

For the Column Type Annotation task, we use two additional small scale datasets, namely, T2D[3] and Efthymiou. We follow the same experimental setting as introduced in Chen et al. (2019b). For this task, the type labels belong to Freebase for the WT-TURL-CT dataset and DBPedia for the T2D and Efthymiou datasets.

For the zero-shot Column Type Annotation (zs-CT) Task, we rearranged the WT-TURL-CT dataset into a new dataset called the zs-WT-TURL-CT dataset. We ensure that individual columns within the train, valid, and test folds do not overlap within this new dataset. For evaluation, we use the trained model to generate a ranked list $\mathcal{L}_c$ of type labels for a given column q_c and evaluate based on the Mean Reciprocal Rank (MRR) and Hits@1 scoring metrics. As q_c can have more than one true label, we use the best-ranked label for evaluation.

For the WT-TURL-EL dataset, we use the same set of candidates as used by Deng et al. (2020)[4]. This dataset only contains instances with the ground truth label within the candidate list. Therefore, we focus on Entity Disambiguation only and use MRR and Hits@1 metrics for evaluation. Additionally, for the T2D dataset (Lehmberg et al., 2016), we do not train a separate model. Rather we use our model trained on the WT-TURL-EL dataset and apply it to T2D's test set.

Moreover, following Deng et al. (2020), we use *two* different experimental settings, i.e., *a*) Using Entity Mentions Only, and *b*) Using Extra metadata. Table 2 illustrates a comparison of different data artifacts used by both TURL and our approach, in either of the experimental settings.

The dataset statistics are available in Table 15 of the Appendix. Also, we will release the zs-WT-TURL-CT dataset once the anonymity period ends.

Table Artifact	Only Entity Mentions		Extra metadata	
	TURL	Ours	TURL	Ours
Page Title	✖	✖	✔	✖
Section Title	✖	✖	✔	✖
Table Caption	✖	✖	✔	✖
Column Header	✖	✖	✔	✔
Linked Entity Info	✖	✖	✔	✔

Table 2: A comparison of Table artifacts used by TURL and our approach on the three table interpretation tasks. See Table 1 for an example.

5.1 Results: Column Type Annotation

Table 3 shows the results on the Column Type annotation task for the WT-TURL-CT dataset. The *first* entry illustrates the performance of Sherlock (Hulsebos et al., 2019), a model that uses 1588 features describing statistical properties, character distributions, word embeddings and paragraph vectors of the cell mentions in a column. TURL (Deng et al., 2020) uses a structure-aware Transformer encoder model, and a pretraining/finetuning strategy to attain the current state of the art for this task.

Approach	Macro F1	Micro F1
Sherlock (Only Entity Mentions)	-	0.785
TURL (Only Entity Mentions)	0.628	0.889
Ours (Only Entity Mentions)	**0.716**	**0.903**
TURL (Extra metadata)	0.805	**0.948**
Ours (Extra metadata)	**0.832**	**0.948**

Table 3: Results for the *Column Type Annotation* (CT) task on the WT-TURL-CT dataset. Both TURL and our approach run over a pretrained TinyBERT model.

The *third* row and the *fifth* row in Table 3 illustrates the performance of our approach on this task, which is an original contribution of this work. Our approach substantially outperforms TURL on the Macro F1, under both the settings, i.e., "Only Entity Mentions" and "Extra metadata". In contrast, for Micro F1, our approach generates a statistically significant *gain* of 1.4% (p-value $<$ 1e-3 using McNemar's test) under the former setting, and

[3]https://github.com/alan-turing-institute/SemAIDA
[4]https://github.com/sunlab-osu/TURL

achieves similar performance on the latter setting.

Next, we compare our approach to the TCN model as introduced by Wang et al. (2021). This model learns unique representation for tables by aggregating information from both a single table and across different tables. Moreover, it employs a supervised multi-task training objective for jointly learning column types and relations between column pairs. In comparison, our approach does *not* use a multi-task training objective, and is column-centric, i.e., encodes the query column only for learning the column types.

TCN is evaluated on the intersection of the WT-TURL-CT and the WT-TURL-RE datasets, i.e., columns present in both these datasets are only considered. The intersection yields a *smaller* dataset having 55,318 tables with 201 type labels and 121 relation labels. Table 4 illustrates the results.

Approach	F1-weighted
TaBERT (Yin et al., 2020)	0.895
TURL (Deng et al., 2020)	0.906
TCN (Wang et al., 2021)	0.933
BERT-Base+PI	**0.944**

Table 4: Comparison of weighted F1 values on an intersection of the WT-TURL-CT and the WT-TURL-RE dataset. The results from the first *two* rows are taken from Wang et al. (2021).

Table 5 illustrates a comparison of Accuracy on T2D and Efthymiou datasets. Following Chen et al. (2019b), we train a single model using 70% of the T2D dataset and then score *separately* on both T2D and Efthymiou datasets. We observe that our approach achieves the new state-of-the-art on T2D, whereas on Efthymiou, it is almost similar in performance (*lags* behind by 0.7%) to TURL. On the other hand, when evaluated on entity mentions only, our approach greatly improves over TURL.

Approach	T2D	Efthymiou
HNN+P2Vec (Chen et al., 2019b)	0.966	0.650
TURL + table metadata	0.962	**0.746**
Ours (entity mentions + col headers)	**0.985**	0.739
TURL (entity mentions only)	0.940	0.516
Ours (entity mentions only)	**0.962**	**0.584**

Table 5: Accuracy results on T2D and Efthymiou, following the setting in Chen et al. (2019b). Both the approaches run over a pretrained TinyBERT model.

5.2 Results: Zero shot Column Type Annotation

Table 6 shows the results on Zero shot Column Type Annotation task when the *choice* of both the Transformer encoder models is either *a*) TinyBERT (Jiao et al., 2020), or *b*) BERT-Base (Devlin et al., 2019). In either case, we observe that using our PI Strategy yields a substantially higher performance, compared to without it.

	TinyBERT		BERT-Base	
	With PI	Without PI	With PI	Without PI
MRR	**0.604**	0.506	**0.674**	0.535
Hits@1	**0.402**	0.310	**0.487**	0.346

Table 6: Results on the Zero-shot *Column Type Annotation* (zs-CT) task. All approaches here use only the entity mentions and column headers.

5.3 Results: Relation Extraction

Table 7 shows the results on the Relation Extraction task for the WT-TURL-RE dataset. The *first* block of two rows compares the performance of TURL with our approach when only entity mentions from a given column pair are used. In comparison, the *second* block illustrates relative performance when extra metadata information is also used.

Approach	Macro F1	Micro F1
TURL (Only Entity Mentions)	0.813	0.905
Ours (Only Entity Mentions)	**0.845**	**0.914**
TURL (Extra metadata)	**0.914**	**0.949**
Ours (Extra metadata)	0.906	0.941

Table 7: Results on the supervised Relation extraction (RE) task. Both the approaches run over a pretrained TinyBERT model.

We observe that when using *entity mentions* only, our approach improves over TURL by 3.2% on the Macro F1, whereas using additional table metadata results in an almost similar Macro F1 by both methods. The Micro F1 remains practically identical in either case, i.e., $\pm 1\%$ between both approaches.

Following Table 2, we notice that, unlike TURL, our approach does not use Page Title, Section Tile, nor Table Caption while operating under the "Extra metadata" setting, yet achieves comparable performances to TURL. How do we incorporate this additional contextual information within our approach is something that we leave as future work.

5.4 Results: Entity Linking

Table 8 shows the results on the Entity Linking task for the WT-TURL-EL and the T2D dataset, wherein the *final* row illustrates the performance of our approach.

Method	WT-TURL-EL		T2D	
	MRR	Hits@1	MRR	Hits@1
Wikidata (Lookup)	0.842	0.785	0.931	0.894
TURL	0.901	0.852	0.899	0.848
Ours	**0.949**	**0.918**	**0.949**	**0.917**

Table 8: Results on the *Entity Linking* (EL) task. Since, we use different scoring metrics, we re-evaluate TURL based on the above benchmark datasets.

Note that the Wikidata lookup baseline achieves a higher score than TURL on the T2D dataset, which is in line with the findings reported by Deng et al. (2020). Nevertheless, our approach substantially outperforms over both Wikidata lookup as well as TURL, on both the datasets.

We observe that running this experiment without our Permutation Invariant (PI) strategy yields similar performance compared to using the PI strategy. We encode the cell values across a row, unlike the CT and RE tasks. Each row within the WT-TURL-EL dataset has roughly *three* textual cell values on average, as opposed to *hundreds* for the CT and RE tasks. We believe that due to this small number of values to encode, we cannot maximize the efficacy of the PI strategy for this task compared to CT and RE tasks. Thus we conclude that the PI Strategy does not provide gains over the default approach when the number of permutation invariant cells being encoded is small.

6 Analysis: Encoding the entire Table

In this section, we compare our column-centric approach against DODUO (Suhara et al., 2021) on the CT task. The DODUO model encodes the entire table and implements a multi-task learning objective for jointly learning column types and relations between column pairs. Table 9 illustrates the comparison results.

The DOSOLO model encodes the entire table, whereas the DOSOLO† model encodes the query column only. In comparison, our approach is column-centric, i.e., it does *not* encode the entire table. Moreover, unlike DODUO, the last *three* rows in Table 9 are finetuned only on the CT Task.

Approach	Encode full Table	Multi-task Learning	Micro F1
DODUO	✔	✔	**0.925**
DOSOLO	✔	✖	0.914
DOSOLO†	✖	✖	0.825
Our Approach	✖	✖	0.920

Table 9: Comparison with existing approaches that encode the entire Table (on the CT Task). All the experiments use entity mentions only as illustrated in Table 2, and fine-tune over a BERT-Base model.

We observe that our approach substantially outperforms DOSOLO†, wherein both these approaches are evaluated under similar experimental settings. Moreover, our method performs slightly below DODUO, even though the latter encodes the entire table as additional context information and performs multi-task learning. These results demonstrate the effectiveness of our approach and open up the possibility of employing a multi-task learning strategy within our method, which we leave as future work.

7 Analysis: Ablation Tests

Table 10 illustrates the ablation experiments performed on the supervised CT and RE tasks respectively. The *first* row in the top block corresponds to using the PI Strategy for the Position IDs. Replacing PI Strategy with the default approach (Section 3) yields the *second* row. In the *third* row, we donot use any Position IDs akin to the SetTransformer model (Lee et al., 2019), whereas the *fourth* row uses a constant Position ID of zero throughout.

The results indicate that *removing* the Position ID parameter altogether performs the same as the default approach. We expected it to be the case, since there is no notion of an ordering between cell-values across a single column. Nevertheless, using our proposed permutation invariant strategy results in the best overall performance.

We also ran additional experiments wherein we created *five* random permutations of the test set corresponding to the supervised CT task. Within each newly created test set, the cell values within each column were randomly permuted and then scored against both trained default TinyBERT and TinyBERT+PI models. Compared to a constant macro f1 score of 0.832 for each of the five tries by the TinyBERT+PI model, the default TinyBERT model had a macro f1 score of 0.81 with a standard devi-

Approach (using Extra metadata)	Macro F1
TinyBERT+PI strategy for Position IDs	**0.832**
TinyBERT (Default approach)	0.812
TinyBERT+No Position ID	0.811
TinyBERT+Constant Position IDs	0.816
TinyBERT (Default Approach)	0.807
TinyBERT+PI strategy for Position IDs	**0.845**

Table 10: Ablation tests on the CT task (*top* block) and RE task (*bottom* block) when evaluated on the WT-TURL-CT and WT-TURL-RE datasets respectively.

ation of 8e-4. These results further reinforce our hypothesis that the performance gains are indeed due to PI.

For the default approach to mirror the PI results, one needs to augment the training dataset with additional shuffled training instances. Such a strategy is not optimal as it increases the runtime complexity for training. In comparison, the PI approach learns a consistent model regardless of the ordering of the cells within a column, thereby demonstrating its efficacy.

The experimental settings, hyper-parameters for our experiments, error analysis, and additional ablation experiments are described in the Appendix.

8 Conclusion

This paper introduces a Permutation Invariant position encoding strategy for transformer-based language models that can encode a set of textual instances independent of its ordering. We argued that such a strategy generates meaningful representations for the rows/columns in a relational table. We show that our proposed method is flexible enough to apply to many downstream tasks. Unlike existing state-of-the-art approaches on table interpretation tasks, our method yields models that do not require additional pretraining yet achieve a new state-of-the-art on almost all the benchmarks.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Jiaoyan Chen, Ernesto Jiménez-Ruiz, Ian Horrocks, and Charles Sutton. 2019a. Colnet: Embedding the semantics of web tables for column type prediction. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 29–36. AAAI Press.

Jiaoyan Chen, Ernesto Jiménez-Ruiz, Ian Horrocks, and Charles Sutton. 2019b. Learning semantic annotations for tabular data. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 2088–2094. ijcai.org.

Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. TURL: table understanding through representation learning. *Proc. VLDB Endow.*, 14(3):307–319.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Harrison Edwards and Amos J. Storkey. 2017. Towards a neural statistician. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Vasilis Efthymiou, Oktie Hassanzadeh, Mariano Rodriguez-Muro, and Vassilis Christophides. 2017. Matching web tables with knowledge base entities: From entity lookups to entity embeddings. In *The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part I*, volume 10587 of *Lecture Notes in Computer Science*, pages 260–277. Springer.

Julian Eisenschlos, Maharshi Gor, Thomas Müller, and William W. Cohen. 2021. MATE: multi-view attention for table transformer efficiency. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 7606–7619. Association for Computational Linguistics.

Michael R. Glass, Mustafa Canim, Alfio Gliozzo, Saneem A. Chemmengath, Vishwajeet Kumar, Rishav Chakravarti, Avi Sil, Feifei Pan, Samarth Bharadwaj, and Nicolas Rodolfo Fauceglia. 2021. Capturing row and column semantics in transformer based question answering over tables. In *Proceedings of the 2021*

Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021, pages 1212–1224. Association for Computational Linguistics.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. Tapas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 4320–4333. Association for Computational Linguistics.

Madelon Hulsebos, Kevin Zeng Hu, Michiel A. Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César A. Hidalgo. 2019. Sherlock: A deep learning approach to semantic data type detection. *CoRR*, abs/1905.10688.

Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. 2021. TABBIE: pretrained representations of tabular data. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 3446–3456. Association for Computational Linguistics.

Maximilian Ilse, Jakub M. Tomczak, and Max Welling. 2018. Attention-based deep multiple instance learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 2132–2141. PMLR.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. Tinybert: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 4163–4174. Association for Computational Linguistics.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. 2019. Set transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3744–3753. PMLR.

Oliver Lehmberg, Dominique Ritze, Robert Meusel, and Christian Bizer. 2016. A large public corpus of web tables containing time and context metadata. In *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11-15, 2016, Companion Volume*, pages 75–76. ACM.

Varish Mulwad, Tim Finin, Zareen Syed, and Anupam Joshi. 2010. Using linked data to interpret tables. In *Proceedings of the First International Workshop on Consuming Linked Data, Shanghai, China, November 8, 2010*, volume 665 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Dominique Ritze, Oliver Lehmberg, and Christian Bizer. 2015. Matching HTML tables to dbpedia. In *Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics, WIMS 2015, Larnaca, Cyprus, July 13-15, 2015*, pages 10:1–10:6. ACM.

Rita Sallam and Carlie Idoine. 2019. Augmented analytics is the future of analytics. *Gartner*.

Adam Santoro, David Raposo, David G. T. Barrett, Mateusz Malinowski, Razvan Pascanu, Peter W. Battaglia, and Tim Lillicrap. 2017. A simple neural network module for relational reasoning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 4967–4976.

Livio Baldini Soares, Nicholas FitzGerald, Jeffrey Ling, and Tom Kwiatkowski. 2019. Matching the blanks: Distributional similarity for relation learning. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 2895–2905. Association for Computational Linguistics.

Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. 2021. Annotating columns with pre-trained language models. *arXiv preprint arXiv:2104.01785*.

Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. 2016. Order matters: Sequence to sequence for sets. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Daheng Wang, Prashant Shiralkar, Colin Lockard, Binxuan Huang, Xin Luna Dong, and Meng Jiang. 2021. TCN: table convolutional network for web table interpretation. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 4020–4032. ACM / IW3C2.

Bo Yang, Sen Wang, Andrew Markham, and Niki Trigoni. 2020. Robust attentional aggregation of deep feature sets for multi-view 3d reconstruction. *Int. J. Comput. Vis.*, 128(1):53–73.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. Tabert: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 8413–8426. Association for Computational Linguistics.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. 2017. Deep sets. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 3391–3401.

Shuo Zhang and Krisztian Balog. 2018. Ad hoc table retrieval using semantic similarity. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pages 1553–1562. ACM.

Shuo Zhang and Krisztian Balog. 2020. Web table extraction, retrieval, and augmentation: A survey. *ACM Trans. Intell. Syst. Technol.*, 11(2):13:1–13:35.

Ziqi Zhang. 2017. Effective and efficient semantic table interpretation using tableminer$^+$. *Semantic Web*, 8(6):921–957.

A Appendix

A.1 Hyperparameters

This section enumerates the experimental details and the hyperparameters used in all of our experiments. For all the methods, we do a grid search over the hyper-parameter space using the validation set, and report the test set results corresponding to the best performing settings. For choosing a threshold value to compute the Macro/Micro F1 metrics, we employ a search space of [0.05, 1.0] with a step size of 0.05. For the learning rate, we performed a grid search over the following values {2e-5, 3e-5, 5e-5}.

For the WT-TURL dataset, the supervised CTA task uses a learning rate of 2e-5. In contrast, the supervised RE task uses a value of 5e-5. These tasks use a single CPU and a GPU, a batch size of 32, are evaluated for a max of 40 epochs, and have early stopping criteria of 5 epochs. The CTA task on T2D and Efthymiou datasets follow the same experimental configuration as the WT-TURL dataset, except that it runs for a max of 100 epochs and has early stopping criteria of 20 epochs.

Similarly, the zero-shot CTA (zs-CTA) and the Entity linking tasks also follow the same experimental configuration as the CTA task, except for the following. The transformer models for this task use a max tokenizer length of 128. The zs-CTA task on TinyBERT/BERT-Base uses eight/sixteen x86 CPUs for concurrent data loading, whereas the Entity linking task uses eight x86 CPUs for the same purpose. Moreover, the zs-CTA/Entity linking task on TinyBERT uses a single GPU for training (with a batch size of 32), whereas on BERT-Base, it uses four GPUs with a combined batch size of 24 for faster training.

We used PyTorch v1.8.1+cu102 for running our experiments, using Intel x86 CPU and Nvidia v100 GPU machines. All of our experiments used a random seed of 73.

A.2 Comparison of Model sizes

Table 11 illustrates a comparison of model sizes in terms of the number of *trainable* parameters of TURL against our proposed approach.

Approach	Additional Pretraining	Finetuning		
		CT	RE	EL
TURL	303 M	14.6 M	14.6 M	14.9 M
Ours	N/A	14.4 M	14.4 M	28.7 M

Table 11: Comparing the model sizes in terms of number of *trainable* parameters (M stands for *millions*).

The TURL method uses a pretrained TinyBERT model. It performs a pretraining step, to begin with, wherein it learns a total of 303 *million* parameters[5]. Out of these, 289 *million* are kept fixed during finetuning. In comparison, we do not perform additional pretraining for our approach. Moreover, our approach requires roughly 200K *less* trainable parameters for fine-tuning on the CT and RE tasks. For the EL task, we employ a siamese network architecture, therefore our approach uses roughly 2x trainable parameters compared to TURL. However, the overall number of model parameters, involving both pre-training and fine-tuning, required by our approach is one order of magnitude less.

A.3 Using Larger Transformer Encoder Models

Our PI Strategy also yields a performance improvement when evaluated against BERT-Base and

[5]Used PyTorch's numel method on trained models available at `https://github.com/sunlab-osu/TURL`

BERT-Large models (Table 12).

Approach	TinyBERT	BERT-Base	BERT-Large
Default	0.812	0.823	0.832
With PI	**0.832**	**0.836**	**0.833**
Params	14M	109M	335M
Time/Epoch	2.5K	18K	69K

Table 12: Ablation tests on the supervised CT task showing Macro F1 when evaluated on the WT-TURL-CT dataset. Training using BERT-Large is done for *three* epochs only. All experiments use Extra metadata as illustrated in Table 2 and are done using one v100 GPU. Furthermore, the Time/Epoch row is rounded off to the nearest *thousands* and is measured in seconds.

As we move from TinyBERT to BERT-Base to BERT-Large, we observe that the number of trainable parameters increases by an overall factor of 24x (comparing TinyBERT to BERT-Large). This increase in the model size is why we believe that the performance improvement due to the PI strategy decreases from 2% (for TinyBERT) to 0.1% (for BERT-Large). Or, in other words, BERT-Large has too many parameters that the learner can finetune, which effectively offsets any gain in performance obtained due to the PI strategy. On the other hand, training BERT-Large is quite cumbersome, i.e., it takes at least one order of magnitude longer to train compared to TinyBERT and is computationally (and environmentally) expensive.

Since TinyBERT is smaller in size, faster in inference, and competitive in performance compared to BERT-Base, a higher performance improvement due to the usage of PI Strategy is *highly significant* in the context of building and deploying practical systems on real-world use cases.

A.4 Error Analysis

In this section, we perform error analysis for the Column Type Annotation and the Relation Extraction tasks. The goal is to identify the *type labels* that are most difficult to predict using our approach. Note that, for this analysis, we use PI Strategy over a TinyBERT model. In addition, we also use column headers and canonical labels corresponding to the linked entity associated with the table cell mention. We selected the top five type labels sorted by the sum of the false positives and false negatives on the test set. Table 13 illustrates these results.

The fine grained types such as *sports.pro_athlete* and *government.politician* in Table 13 have a classification error rate of 26.5% and 25.3% respectively, whereas coarse-grained types such as people.person have a 2.5% error rate. Furthermore, out of 138 false positives for the label *location.citytown*, 135 of them have *location.location* as a true label. On the other hand, all 89 test instances that get marked as False negatives (for *location.citytown*) have *location.location* as one of the ground truth labels, and we can identify 74 of them correctly. Table 14 illustrates the result of a similar analysis on the Relation Extraction task.

Type label	Test counts	False neg	False pos
sports.pro_athlete	1688	129	319
organization.organization	2122	121	156
location.citytown	562	89	138
location.location	3245	80	50
government.politician	495	73	52

Table 13: False negative/positive type predictions on the WT-TURL-CT dataset.

If we *generalize* these observations, we can conclude that our approach *a*) Makes lot more errors on fine-grained type labels as opposed to coarse–grained labels, and *b*) Confuses often between labels that are semantically related. This behavior is expected since it is difficult for any model to resolve these issues based only on tabular data, i.e., without additional table contextual information. As mentioned before, we leave the strategy of inculcating contextual information into our learning approach for future work.

Relation Label	False Neg	False Pos
award.award_nominated_work.award_nominations-award.award_nomination.award_nominee	37	29
award.award_winning_work.awards_won-award.award_honor.award_winner	23	19
film.film.written_by	16	10

Table 14: False negative/positive relation predictions on the WT-TURL-RE dataset. Out of 37 counts of false negatives for the *first* relation label, our approach predicts *film.film.directed_by* (23 counts), *film.film.starring-film.performance.actor* (7 counts), and *film.film.written_by* (7 counts). Similarly, out of 23 counts of false negatives for the *second* relation label, our approach predicts *award.award_nominated_work.award_nominations-award.award_nomination.award_nominee* (12 counts) and *film.film.written_by* (5 counts). For all of these relation labels, the corresponding test instances contained *two* columns, one containing names of people, and the other containing names of movies, songs, albums, etc. Since, our approach does not use table context information, resolving these label confusions seems difficult.

Task Name	Dataset	Train	Dev	Test	Target Labels
CT Task	WT-TURL-CT	628,254	13,391	13,025	255
	T2D	250	-	133	37
	Efthymiou	250	-	620	37
zs-CT Task	zs-WT-TURL-CT	409,125 (190)	106,756 (24)	29,259 (25)	-
RE Task	WT-TURL-RE	62,954	2,175	2,072	121
EL Task	WT-TURL-EL	1,264,217	76,720	225,777	-
	T2D	-	-	21,270	-

Table 15: Dataset statistics for the Column Type Annotation (CT), Zero-Shot Column Type Annotation (zs-CT), Relation Extraction (RE), and Entity Linking (EL) Tasks. For the zs-CT task, the number within the *brackets* (for the train/dev/test folds) indicates the number of class labels available for that fold. For the EL task, we have a total of roughly 5 *million* candidate entities due to Wikidata lookup across all the instances.