TURL: Table Understanding through Representation Learning

Xiang Deng*
The Ohio State University
Columbus, Ohio
deng.595@buckeyemail.osu.edu

Huan Sun*
The Ohio State University
Columbus, Ohio
sun.397@osu.edu

Alyssa Lees
Google Research
New York, NY
alyssalees@google.com

You Wu
Google Research
New York, NY
wuyou@google.com

Cong Yu
Google Research
New York, NY
congyu@google.com

ABSTRACT

Relational tables on the Web store a vast amount of knowledge. Owing to the wealth of such tables, there has been tremendous progress on a variety of tasks in the area of table understanding. However, existing work generally relies on heavily-engineered task-specific features and model architectures. In this paper, we present TURL, a novel framework that introduces the pre-training/fine-tuning paradigm to relational Web tables. During pre-training, our framework learns deep contextualized representations on relational tables in an unsupervised manner. Its universal model design with pre-trained representations can be applied to a wide range of tasks with minimal task-specific fine-tuning.

Specifically, we propose a structure-aware Transformer encoder to model the row-column structure of relational tables, and present a new Masked Entity Recovery (MER) objective for pre-training to capture the semantics and knowledge in large-scale unlabeled data. We systematically evaluate TURL with a benchmark consisting of 6 different tasks for table understanding (e.g., relation extraction, cell filling). We show that TURL generalizes well to all tasks and substantially outperforms existing methods in almost all instances.

PVLDB Reference Format:
Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. TURL: Table Understanding through Representation Learning. PVLDB, 14(3): 307 - 319, 2021.
doi:10.14778/3430915.3430921

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/sunlab-osu/TURL.

1 INTRODUCTION

Relational tables are in abundance on the Web and store a large amount of knowledge, often with key entities in one column and attributes in the others. Over the past decade, various large-scale collections of such tables have been aggregated [4, 6, 7, 26]. For example, Cafarella et al. [6, 7] reported 154M relational tables out of a total of 14.1 billion tables in 2008. More recently, Bhagavatula et al. [4] extracted 1.6M high-quality relational tables from Wikipedia. Owing to the wealth and utility of these datasets, various tasks such as table interpretation [4, 17, 30, 36, 48, 49], table augmentation [1, 6, 12, 43, 46, 47], etc., have made tremendous progress in the past few years.

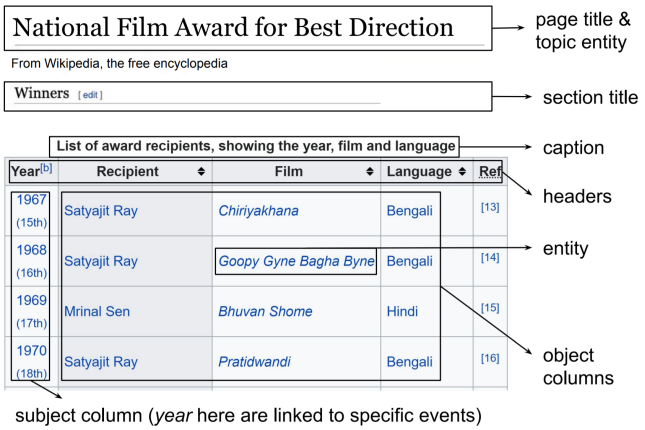


Figure 1: An example of a relational table from Wikipedia.

However, previous work such as [4, 46, 47] often rely on heavily-engineered task-specific methods such as simple statistical/language features or straightforward string matching. These techniques suffer from several disadvantages. First, simple features only capture shallow patterns and often fail to handle the flexible schema and varied expressions in Web tables. Second, task-specific features and model architectures require effort to design and do not generalize well across tasks.

Recently, the pre-training/fine-tuning paradigm has achieved notable success on unstructured text data. Advanced language models such as BERT [16] can be pre-trained on large-scale unsupervised text and subsequently fine-tuned on downstream tasks using task-specific supervision. In contrast, little effort has been extended to the study of such paradigms on relational tables. Our work fills this research gap.

Promising results in some table based tasks were achieved by the representation learning model of [13]. This work serializes a table into a sequence of words and entities (similar to text data) and learns embedding vectors for words and entities using Word2Vec [29]. However, [13] cannot generate *contextualized* representations, i.e., it does not consider varied use of words/entities in different contexts

*Corresponding authors.

Proceedings of the VLDB Endowment, Vol. 14, No. 3 ISSN 2150-8097.
doi:10.14778/3430915.3430921

and only produces a single fixed embedding vector for word/entity. In addition, *shallow* neural models like Word2Vec have relatively limited learning capabilities, which hinder the capture of complex semantic knowledge contained in relational tables.

We propose TURL, a novel framework for learning deep contextualized representations on relational tables via pre-training in an unsupervised manner and task-specific fine-tuning.

There are two main challenges in the development of TURL: (1) *Relational table encoding*. Existing neural network encoders are designed for linearized sequence input and are a good fit with unstructured texts. However, data in relational tables is organized in a semi-structured format. Moreover, a relational table contains multiple components including the table caption, headers and cell values. The challenge is to develop a means of modeling the row-and-column structure as well as integrating the heterogeneous information from different components of the table. (2) *Factual knowledge modeling*. Pre-trained language models like BERT [16] and ELMo [32] focus on modeling the syntactic and semantic characteristics of word use in natural sentences. However, relational tables contain a vast amount of factual knowledge about entities, which cannot be captured by existing language models directly. Effectively modelling such knowledge in TURL is a second challenge.

To address the first challenge, we encode information from different table components into separate input embeddings and fuse them together. We then employ a *structure-aware* Transformer [39] encoder with masked self-attention. The conventional Transformer model is a bi-directional encoder, so each element (i.e., token/entity) can attend to all other elements in the sequence. We explicitly model the row-and-column structure by restraining each element to only aggregate information from other structurally related elements. To achieve this, we build a visibility matrix based on the table structure and use it as an additional mask for the self-attention layer.

For the second challenge, we first learn embeddings for each entity during pre-training. We then model the relation between entities in the same row or column with the assistance of the visibility matrix. Finally, we propose a Masked Entity Recovery (MER) pre-training objective. The technique randomly masks out entities in a table with the objective of recovering the masked items based on other entities and the table context (e.g., caption/header). *This encourages the model to learn factual knowledge from the tables and encode it into entity embeddings.* In addition, we utilize the entity mention by keeping it as additional information for a certain percentage of masked entities. *This helps our model build connections between words and entities* . We also adopt the Masked Language Model (MLM) objective from BERT, which aims to model the complex characteristics of word use in table metadata.

We pre-train our model on 570K relational tables from Wikipedia to generate contextualized representations for tokens and entities in the relational tables. We then fine-tune our model for specific downstream tasks using task-specific labeled data. A distinguishing feature of TURL is its universal architecture across different tasks - only minimal modification is needed to cope with each downstream task. To facilitate research in this direction, we compiled a benchmark that consists of 6 diverse tasks, including entity linking, column type annotation, relation extraction, row population, cell filling and schema augmentation. We created new datasets in addition to including results for existing datasets when publicly available. Experimental results show that TURL substantially outperforms existing task-specific and shallow Word2Vec based methods.

Table 1: Summary of notations for our table data.

Symbol	Description
T	A relational table $T = (C, H, E, e_t)$
C	Table caption (a sequence of tokens)
H	Table schema $H = \{h_0, ..., h_i, ..., h_m\}$
h_i	A column header (a sequence of tokens)
E	Columns in table that contains entities
e_t	The topic entity of the table $e_t = (e_t^{\mathrm{e}}, e_t^{\mathrm{m}})$
e	An entity cell $e = (e^{\mathrm{e}}, e^{\mathrm{m}})$

Our contributions are summarized as follows:

- To the best of our knowledge, TURL is the first framework that introduces the pre-training/fine-tuning paradigm to relational Web tables. The pre-trained representations along with the universal model design save tremendous effort on engineering task-specific features and architectures.
- We propose a structure-aware Transformer encoder to model the structure information in relational tables. We also present a novel Masked Entity Recovery (MER) pre-training objective to learn the semantics as well as the factual knowledge about entities in relational tables.
- To facilitate research in this direction, we present a benchmark that consists of 6 different tasks for table interpretation and augmentation. We show that TURL generalizes well to various tasks and substantially outperforms existing models. Our source code, benchmark, as well as pre-trained models will be available online.

2 PRELIMINARY

We now present our data model and give a formal task definition:

In this work, we focus on relational Web tables and are most interested in the factual knowledge about entities. Each table $T \in \mathcal{T}$ is associated with the following: (1) Table caption C, which is a short text description summarizing what the table is about. When the page title or section title of a table is available, we concatenate these with the table caption. (2) Table headers H, which define the table schema; (3) Topic entity e_t, which describes what the table is about and is usually extracted from the table caption or page title; (4) Table cells E containing entities. Each entity cell $e \in E$ contains a specific object with a unique identifier. For each cell, we define the entity as $e = (e^{\mathrm{e}}, e^{\mathrm{m}})$, where e^{e} is the specific entity linked to the cell and e^{m} is the entity mention (i.e., the text string).

(C, H, e_t) is also known as *table metadata* and E is the actual *table content*. Notations used in the data model are summarized in Table 1.

Explicitly, we study the unsupervised representation learning on relational Web tables, which is defined as follows.

Definition 2.1. Given a relational Web table corpus, our representation learning task aims to learn in an unsupervised manner a task-agnostic contextualized vector representation for each token in all table captions C's and headers H's and for each entity (i.e., all entity cells E's and topic entities e_t's).

3 RELATED WORK

Representation Learning. The pre-training/fine-tuning paradigm has drawn tremendous attention in recent years. Extensive effort has been devoted to the development of unsupervised representation learning methods for both unstructured text and structured knowledge bases, which in turn can be utilized for a wide variety of downstream tasks via fine-tuning.

Earlier work, including Word2Vec [29] and GloVe [31], pre-train distributed representations for words on large collections of documents. The resulting representations are widely used as input embeddings and offer significant improvements over randomly initialized parameters. However, pre-trained word embeddings suffer from word polysemy: they cannot model varied word use across linguistic contexts. This complexity motivated the development of contextualized word representations [16, 32, 44] . Instead of learning fixed embeddings per word, these works construct language models that learn the joint probabilities of sentences. Such pre-trained language models have had huge success and yield state-of-the-art results on various NLP tasks [40].

Similarly, unsupervised representation learning has also been adopted in the space of structured data like knowledge bases (KB) and databases. Entities and relations in KB have been embedded into continuous vector spaces that still preserve the inherent structure of the KB [38]. These entity and relation embeddings are utilized by a variety of tasks, such as KB completion [5, 41], relation extraction [35, 42], entity resolution [19], etc. Similarly, [18] learned embeddings for heterogeneous data in databases and used it for data integration tasks.

More recently, there has been a corpus of work incorporating knowledge information into pre-trained language models [33, 50]. ERNIE [50] injects knowledge base information into a pre-trained BERT model by utilizing pre-trained KB embeddings and a denoising entity autoencoder objective. The experimental results demonstrate that knowledge information is extremely helpful for tasks such as entity linking, entity typing and relation extraction.

Despite the success of representation learning on text and KB, few works have thoroughly explored contextualized representation learning on relational Web tables. Pre-trained language models are directly adopted in [27] for entity matching. Two recent papers from the NLP community [21, 45] study pre-training on Web tables to assist in semantic parsing or question answering tasks on tables. In this work, we introduce TURL, a new methodology for learning deep contextualized representations for relational Web tables that preserve both semantic and knowledge information. In addition, we conduct comprehensive experiments on a much wider range of table-related tasks.

Table Interpretation. The Web stores large amounts of knowledge in relational tables. Table interpretation aims to uncover the semantic attributes of the data contained in relational tables, and transform this information into machine understandable knowledge. This task is usually accomplished with help from existing knowledge bases. In turn, the extracted knowledge can be used for KB construction and population.

There are three main tasks for table interpretation: entity linking, column type annotation and relation extraction [4, 48]. Entity linking is the task of detecting and disambiguating specific entities mentioned in a table. Since relational tables are centered around entities, entity linking is a key step for table interpretation, and a fundamental component to many table-related tasks [48]. [4] employed a graphical model, and used a collective classification technique to optimize a global coherence score for a set of entities in a table. [36] presented the T2K framework, which is an iterative matching approach that combines both schema and entity matching. More recently, [17] introduced a hybrid method that combines both entity lookup and entity embeddings, which resulted in superior performance on various benchmarks.

Column type annotation and relation extraction both work with table columns. The former aims to annotate columns with KB types while the latter intends to use KB predicates to interpret relations between column pairs. Prior work has generally coupled these two tasks with entity linking [30, 36, 49]. After linking cells to entities, the types and relations associated with the entities in KB can then be used to annotate columns. In recent work, column annotation without entity linking has been explored [10, 11, 22]. These works modify text classification models to fit relational tables and have shown promising results. Moreover, relation extraction on web tables has also been studied for KB augmentation [8, 14, 37].

Table Augmentation. Tables are a popular data format to organize and present relational information. Users often have to manually compose tables when gathering information. It is desirable to offer some intelligent assistance to the user, which motivates the study of table augmentation [46]. Table augmentation refers to the task of expanding a seed query table with additional data. Specifically, for relational tables this can be divided into three sub-tasks: row population for retrieving entities for the subject column [12, 46], cell filling that fills the cell values for given subject entities [1, 43, 47] and schema augmentation that recommends headers to complete the table schema [6, 46]. For row population tasks, [12] searches for complement tables that are semantically related to seed entities and the top ranked tables are used for population. [46] further incorporates knowledge base information with a table corpus, and develops a generative probabilistic model to rank candidate entities with entity similarity features. For cell filling, [43] uses the query table to search for matching tables, and extracts attribute values from those tables. More recently, [47] proposed the CellAutoComplete framework that makes use of a large table corpus and a knowledge base as data sources, and incorporates preprocessing, candidate value finding, and value ranking components. In terms of schema augmentation, [6] tackles this problem by utilizing an attribute correlation statistics database (ACSDb) collected from a table corpus. [46] utilizes a similar approach to the row population techniques and ranks candidate headers with sets of features.

Existing benchmarks. Several benchmarks have been proposed for table interpretation: (1) T2Dv2 [26] proposed in 2016 contains 779 tables from various websites. It contains 546 relational tables, with 25119 entity annotations, 237 table-to-class annotations and 618 attribute-to-property annotations. (2) Limaye et al. [28] proposed a benchmark in 2010 which contains 296 tables from Wikipedia. It was used in [17] for entity linking, and was also used in [11] for column type annotation. (3) Efthymiou et al. [17] created a benchmark (referred to as "WikiGS" in our experiments) that includes 485,096 tables from Wikipedia. WikiGS was originally

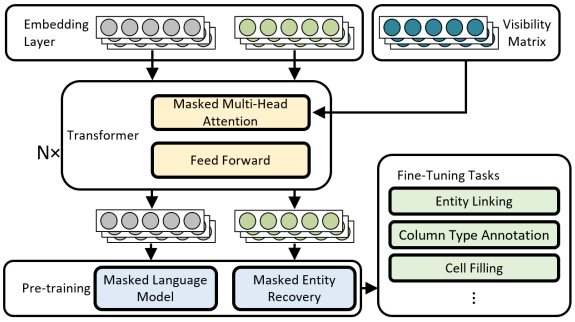


Figure 2: Overview of our TURL framework.

used for entity linking with 4,453,329 entity matches. [11] further annotated a subset of it containing 620 entity columns with 31 DBpedia types and used it for column type annotation. (4) The recent SemTab 2019 [24] challenge also aims at benchmarking systems that match tabular data to KBs, including three tasks, i.e., assigning a semantic type to an column, matching a cell to an entity, and assigning a property to the relationship between two columns. It used sampled tables from T2Dv2 [26] and WikiGS [17] in the first two rounds, and automatically generated tables in later rounds.

In contrast to table interpretation, few benchmarks have been released for table augmentation. Zhang et al. [46] studied row population and schema augmentation with 2000 randomly sampled Wikipedia tables in total for validation and testing. [47] curated a test collection with 200 columns containing 1000 cells from Wikipedia tables for evaluating cell filling.

Although these benchmarks have been used in various recent studies, they still suffer from a few shortcomings: (1) They are typically small sets of sampled tables with limited annotations. (2) SemTab 2019 contains a large number of instances; however, most of them are automatically generated and lack metadata/context of the Web tables. In this work, we compile a larger benchmark covering both table interpretation and table augmentation tasks. We also use some of these existing datasets for more comprehensive evaluation. By leveraging large-scale relational tables on Wikipedia and a curated KB, we ensure both the size and quality of our dataset.

4 METHODOLOGY

In this section, we introduce our TURL framework for unsupervised representation learning on relational tables. TURL is first trained on an unlabeled relational Web table corpus with pre-training objectives carefully designed to learn word semantics as well as relational knowledge between entities. The model architecture is general and can be applied to a wide range of downstream tasks with minimal modifications. Moreover, the pre-training process alleviates the need for large-scale labeled data for each downstream task.

4.1 Model Architecture

Figure 2 presents an overview of TURL which consists of three modules: (1) an embedding layer to convert different components of an input table into input embeddings, (2) N stacked structure-aware Transformer [39] encoders to capture the textual information and relational knowledge, and (3) a final projection layer for pre-training objectives. Figure 3 shows an input-output example.

4.2 Embedding Layer

Given a table T=(C, H, E, e_t), we first linearize the input into a sequence of tokens and entity cells by concatenating the table metadata and scanning the table content row by row. The embedding layer then converts each token in C and H and each entity in E and e_t into an embedding representation.

Input token representation. For each token w, its vector representation is obtained as follows:

$$\mathbf{x}^{\mathrm{t}} = \mathbf{w} + \mathbf{t} + \mathbf{p}. \tag{1}$$

Here $\mathbf{w}$ is the word embedding vector, $\mathbf{t}$ is called the type embedding vector and aims to differentiate whether token w is in the table caption or a header, and $\mathbf{p}$ is the position embedding vector that provides relative position information for a token within the caption or a header.

Input entity representation. For each entity cell $e = (e^{\mathrm{e}}, e^{\mathrm{m}})$ (same for topic entity e_t), we fuse the information from the linked entity e^{e} and entity mention e^{m} together, and use an additional type embedding vector $\mathbf{t}^{\mathrm{e}}$ to differentiate three types of entity cells (i.e., subject/object/topic entities). Specifically, we calculate the input entity representation $\mathbf{x}^{\mathrm{e}}$ as:

$$\mathbf{x}^{\mathrm{e}} = \texttt{LINEAR}([\mathbf{e}^{\mathrm{e}}; \mathbf{e}^{\mathrm{m}}]) + \mathbf{t}^{\mathrm{e}}; \tag{2}$$

$$\mathbf{e}^{\mathrm{m}} = \texttt{MEAN}(\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_j, \ldots). \tag{3}$$

Here $\mathbf{e}^{\mathrm{e}}$ is the entity embedding learned during pre-training. To represent entity mention e^{m}, we use its average word embedding $\mathbf{w}_j$'s. LINEAR is a linear layer to fuse $\mathbf{e}^{\mathrm{e}}$ and $\mathbf{e}^{\mathrm{m}}$.

A sequence of token and entity representations ($\mathbf{x}^{\mathrm{t}}$'s and $\mathbf{x}^{\mathrm{e}}$'s) are then fed into the next module of TURL, a structure-aware Transformer encoder, which will produce contextualized representations.

4.3 Structure-aware Transformer Encoder

We choose Transformer [39] as our base encoder block, since it has been widely used in pre-trained language models [16, 34] and achieves superior performance on various natural language processing tasks [40]. Due to space constraints, we only briefly introduce the conventional Transformer encoder and refer readers to [39] for more details. Finally, we present a detailed explanation on our proposed visibility matrix for modeling table structure.

Each Transformer block is composed of a multi-head self-attention layer followed by a point-wise, fully connected layer [39]. Specifically, we calculate the multi-head attention as follows:

$$\begin{aligned} \mathrm{MultiHead}(\mathbf{h}) &= [\mathrm{head}_1; \ldots; \mathrm{head}_{\mathrm{i}}; \ldots; \mathrm{head}_{\mathrm{k}}]W^O; \\ \mathrm{head}_{\mathrm{i}} &= \mathrm{Attention}\left(\mathbf{h}W_i^Q, \mathbf{h}W_i^K, \mathbf{h}W_i^V\right); \\ \mathrm{Attention}(Q, K, V) &= \mathrm{Softmax}\left(\frac{QK^T}{\sqrt{d}}M\right)V. \end{aligned} \tag{4}$$

Here $\mathbf{h} \in \mathbb{R}^{n \times d_{\mathrm{model}}}$ is the hidden state output from the previous Transformer layer or the input embedding layer and n is the input sequence length. $\frac{1}{\sqrt{d}}$ is the scaling factor. $W_i^Q \in \mathbb{R}^{d_{\mathrm{model}} \times d}$,$W_i^K \in \mathbb{R}^{d_{\mathrm{model}} \times d}$,$W_i^V \in \mathbb{R}^{d_{\mathrm{model}} \times d}$ and $W^O \in \mathbb{R}^{kd \times d_{\mathrm{intermediate}}}$ are parameter matrices. For each head, we have $d = d_{\mathrm{model}}/\mathrm{k}$, where k is the number of attention heads. $M \in \mathbb{R}^{n \times n}$ is the visibility matrix which we detail next.

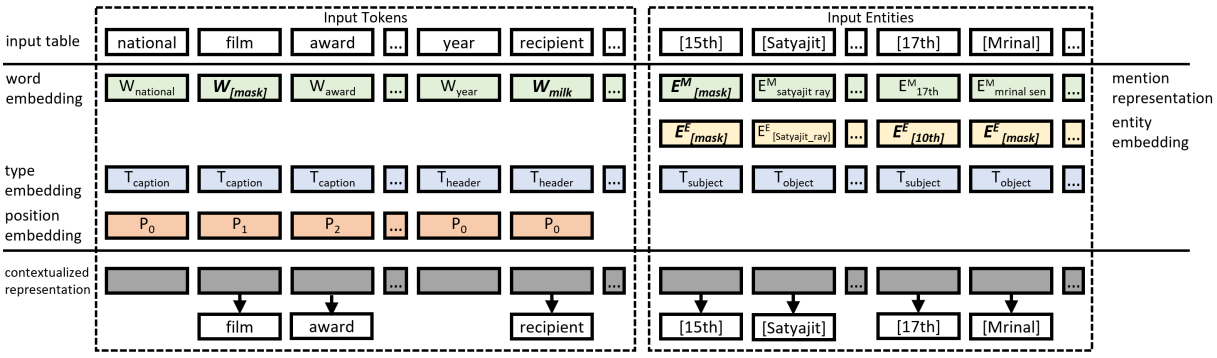


Figure 3: Illustration of the model input-output. The input table is first transformed into a sequence of tokens and entity cells, and processed for structure-aware Transformer encoder as described in Section 4.4. We then get contextualized representations for the table and use them for pre-training. Here [15th] **(which means** 15th National Film Awards**),** [Satyajit]**, ... are linked entity cells.**

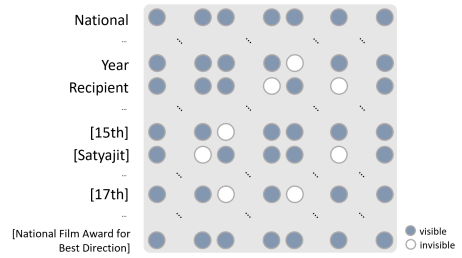


Figure 4: Graphical illustration of visibility matrix (symmetric).

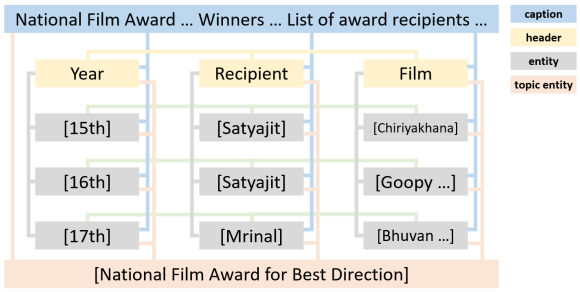


Figure 5: Graphical illustration of masked self-attention by our visibility matrix. Each token/entity in a table can only attend to its directly connected neighbors (shown as edges here).

Visibility matrix. To interpret relational tables and extract the knowledge embedded in them, it is important to model row-column structure. For example, in Figure 1, [Satyajit] and [Chiriyakhana] are related because they are in the same row, which implies that [Satyajit] directs [Chiriyakhana]. In contrast, [Satyajit] should not be related to [Pratidwandi]. Similarly, [Hindi] is a "*language*" and its representation has little to do with the header "*Film*". We propose a visibility matrix M to model such structure information in a relational table. Figure 4 shows an example of M.

Our visibility matrix acts as an attention mask so that each token (or entity) can only aggregate information from other structurally related tokens/entities during the self-attention calculation. M is a symmetric binary matrix with $M_{i,j} = 1$ if and only if $\texttt{element}_j$ is visible to $\texttt{element}_i$. The element here can be a token in the caption or a header, or an entity in a table cell. Specifically, we define M as follows:

- If $\texttt{element}_i$ is the topic entity or a token in table caption, $\forall j, M_{i,j} = 1$. *Table caption and topic entity are visible to all components of the table.*
- If $\texttt{element}_i$ is a token or an entity in the table and $\texttt{element}_j$ is a token or an entity in the same row or the same column, $M_{i,j} = 1$. *Entities and text content in the same row or the same column are visible to each other.*

4.4 Pre-training Objective

In order to pre-train our model on an unlabeled table corpus, we adopt the Masked Language Model (MLM) objective from BERT to learn representations for tokens in table metadata and propose a Masked Entity Recovery (MER) objective to learn entity cell representations.

Masked Language Model. We adopt the same Masked Language Model objective as BERT, which trains the model to capture the lexical, semantic and contextual information described by table metadata. Given an input token sequence including table caption and table headers, we simply mask some percentage of the tokens at random, and then predict these masked tokens. We adopt the same percentage settings as BERT. The pre-training data processor selects 20% of the token positions at random (note, we use a slightly larger ratio compared with 15% in [16] as we want to make the pre-training more challenging). For a selected position, (1) 80% of the time we replace it with a special [MASK] token, (2) 10% of the time we replace it with another random token, and (3) 10% of the time we keep it unchanged.

Example 4.1. Figure 3 shows an example of the above random process in MLM, where (1) "*film*", "*award*" and "*recipient*" are chosen randomly, and (2) the input word embedding of "*film*" is further chosen randomly to be replaced with the embedding of [MASK], (3) the input word embedding of "*recipient*" to be replaced with the embedding of a random word "*milk*", and (4) the input word embedding of "*award*" to remain the same.

Given a token position selected for MLM, which has a contextualized representation $\mathbf{h}^t$ output by our encoder, the probability of predicting its original token $w \in \mathcal{W}$ is then calculated as:

$$P(w) = \frac{\exp\left(\texttt{LINEAR}(\mathbf{h}^t) \cdot \mathbf{w}\right)}{\sum_{w_k \in \mathcal{W}} \exp\left(\texttt{LINEAR}(\mathbf{h}^t) \cdot \mathbf{w}_k\right)} \tag{5}$$

Masked Entity Recovery. In addition to MLM, we propose a novel Masked Entity Recovery (MER) objective to help the model capture the factual knowledge embedded in the table content as well as the associations between table metadata and table content. Essentially, we mask a certain percentage of input entity cells and then recover the linked entity based on surrounding entity cells and table metadata. This requires the model to be able to infer the relation between entities from table metadata and encode the knowledge in entity embeddings.

In addition, our proposed masking mechanism takes advantage of entity mentions. Specifically, as shown in Eqn. 2, the input entity representation has two parts: the entity embedding $\mathbf{e}^{\mathrm{e}}$ and the entity mention representation $\mathbf{e}^{\mathrm{m}}$. For some percentage of masked entity cells, we only mask $\mathbf{e}^{\mathrm{e}}$, and as such the model receives additional entity mention information to help form predictions. This assists the model in building a connection between entity embeddings and entity mentions, and helps downstream tasks where only cell texts are available.

Specifically, we propose the following masking mechanism for MER: The pre-training data processor chooses 60% of entity cells at random. Here we adopt a higher masking ratio for MER compared with MLM, because oftentimes in downstream tasks, none or few entities are given. For one chosen entity cell, (1) 10% of the time we keep both $\mathbf{e}^{\mathrm{m}}$ and $\mathbf{e}^{\mathrm{e}}$ unchanged (2) 63% (i.e., 70% of the left 90%) of the time we mask both $\mathbf{e}^{\mathrm{m}}$ and $\mathbf{e}^{\mathrm{e}}$ (3) 27% (i.e., 30% of the left 90%) of the time we keep $\mathbf{e}^{\mathrm{m}}$ unchanged, and mask $\mathbf{e}^{\mathrm{e}}$ (among which we replace $\mathbf{e}^{\mathrm{e}}$ with embedding of a random entity to inject noise in 10% of the time). Similar to BERT, in both MLM and MER we keep a certain portion of the selected positions unchanged so that the model can generate good representations for non-masked tokens/entity cells. Trained with random tokens/entities replacing the original ones, the model is robust and utilizes contextual information to make predictions rather than simply copying the input representation.

Example 4.2. Take Figure 3 as an example. [15th], [Satyajit], [17th] and [Mrinal] are first chosen for MER. Then, (1) the input mention representation and entity embedding of [Satyajit] remain the same. (2) The input mention representation and entity embedding of [15th] are both replaced with the embedding of [MASK] (3) The input entity embedding of [Mrinal] is replaced with embedding of [MASK], while the input entity embedding of [17th] is replaced with the embedding of a random entity [10th]. In both cases, the input mention representation are unchanged.

Given an entity cell selected for MER with a contextualized representation $\mathbf{h}^{\mathrm{e}}$ output by our encoder, the probability of predicting entity $e \in \mathcal{E}$ is then calculated as follows.

$$P(e) = \frac{\exp\left(\texttt{LINEAR}(\mathbf{h}^{\mathrm{e}}) \cdot \mathbf{e}^{\mathrm{e}}\right)}{\sum_{e_k \in \mathcal{E}} \exp\left(\texttt{LINEAR}(\mathbf{h}^{\mathrm{e}}) \cdot \mathbf{e}^{\mathrm{e}}_k\right)} \tag{6}$$

In reality, considering the entity vocabulary $\mathcal{E}$ is quite large, we only use the above equation to rank entities from a given candidate set. For efficient training, we construct the candidate set with (1) entities in the current table, (2) entities that have co-occurred with those in the current table, and (3) randomly sampled negative entities.

Table 2: Dataset statistics (per table) in pre-training.

	split	min	mean	median	max
# row	train	1	13	8	4670
	dev	5	20	12	667
	test	5	21	12	3143
# ent. columns	train	1	2	2	20
	dev	3	4	3	15
	test	3	4	3	15
# ent.	train	3	19	9	3911
	dev	8	57	34	2132
	test	8	60	34	9215

We use a cross-entropy loss function for both MLM and MER objectives and the final pre-training loss is given as follows:

$$loss = \sum \log\left(P(w)\right) + \sum \log\left(P(e)\right), \tag{7}$$

where the sums are over all tokens and entity cells selected in MLM and MER respectively.

Pre-training details. In this work, we denote the number of Transformer blocks as N, the hidden dimension of input embeddings and all Transformer block outputs as d_{model}, the hidden dimension of the fully connected layer in a Transformer block as $d_{\text{intermediate}}$, and the number of self-attention heads as k. We take advantage of a pre-trained TinyBERT [23] model, which is a knowledge distilled version of BERT with a smaller size, and set the hyperparameters as follows: N = 4, $d_{\text{model}} = 312$, $d_{\text{intermediate}} = 1200$, k = 12. We initialize our structure-aware Transformer encoder parameters, word embeddings and position embeddings with TinyBERT [23]. Entity embeddings are initialized using averaged word embeddings in entity names, and type embeddings are randomly initialized. We use the Adam [25] optimizer with a linearly decreasing learning rate. The initial learning rate is 1e-4 chosen from [1e-3, 5e-4, 1e-4, 1e-5] based on our validation set. We pre-trained the model for 80 epochs. Due to space constraints, we refer readers to the extended version of our paper [15] for detailed ablation studies on the design choices.

5 DATASET CONSTRUCTION FOR PRE-TRAINING

We construct a dataset for unsupervised representation learning based on the WikiTable corpus [4], which originally contains around 1.65M tables extracted from Wikipedia pages. The corpus contains a large amount of factual knowledge on various topics ranging from sport events (e.g., Olympics) to artistic works (e.g., TV series). The following sections introduce our data construction process as well as characteristics of the dataset.

5.1 Data Pre-processing and Partitioning

Pre-processing. The corresponding Wikipedia page of a table often provides much contextual information, such as page title and section title that can aid in the understanding of a table topic. We concatenate page title, section title and table caption to obtain a comprehensive description.

In addition, each table in the corpus contains one or more header rows and several rows of table content. For tables with more than one header row, we concatenate headers in the same column to

obtain one header for each column. For each cell, we obtain hyperlinks to Wikipedia pages in it and use them to normalize different entity mentions corresponding to the same entity. We treat each Wikipedia page as an individual entity and do not use additional tools to perform entity linking with an external KB. For cells containing multiple hyperlinks, we only keep the first link. We also discard rows that have merged columns in a table.

Identify relational tables. We first locate all columns that contain at least one linked cell after pre-processing. We further filter out noisy columns with empty or illegal headers (e.g., *note*, *comment*, *reference*, digit numbers, etc.). The columns left are entity-centric and are referred to as entity columns. We then identify relational tables by finding tables that have a subject column. A simple heuristic is employed for subject column detection: the subject column must be located in the first two columns of the table and contain unique entities which we treat as subject entities. We further filter out tables containing less than three entities or more than twenty columns. With this process, we obtain 670,171 relational tables.

Data partitioning. From the above 670,171 tables, we select a high quality subset for evaluation: From tables that have (1) more than four linked entities in the subject column, (2) at least three entity columns including the subject column, and (3) more than half of the cells in entity columns are linked, we randomly select 10000 to form a held-out set. We further randomly partition this set into validation/test sets via a rough 1:1 ratio for model evaluation. All relational tables not in the evaluation set are used for pre-training. In sum, we have 570171 / 5036 / 4964 tables respectively for pre-training/validation/test sets.

5.2 Dataset Statistics in Pre-training

Fine-grained statistics of our datasets are summarized in Table 2. We can see that most tables in our pre-training dataset have moderate size, with median of 8 rows, 2 entity columns and 9 entities per table. We build a token vocabulary using the BERT-based tokenizer [16] (with 30,522 tokens in total). For the entity vocabulary, we construct it based on the training table corpus and obtain 926,135 entities after removing those that appear only once.

6 EXPERIMENTS

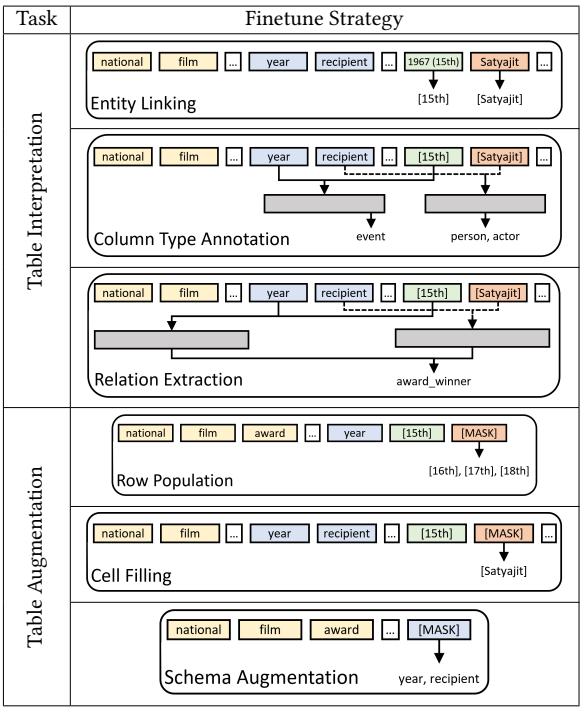
To systematically evaluate our pre-trained framework as well as facilitate research, we compile a table understanding benchmark consisting of 6 widely studied tasks covering table interpretation (e.g., entity linking, column type annotation, relation extraction) and table augmentation (e.g., row population, cell filling, schema augmentation). We include existing datasets for entity linking. However, due to the lack of large-scale open-sourced datasets, we create new datasets for other tasks based on our held-out set of relational tables and an existing KB.

Next we introduce the definition, baselines, dataset and results for each task. Our pre-trained framework is general and can be fine-tuned for all the independent tasks.

6.1 General Setup across All Tasks

We use the pre-training tables to create the training set for each task, and always build data for evaluation using the held-out validation/test tables. This way we ensure that there is no overlapping

Table 3: An overview of our benchmark tasks and strategies to fine-tune TURL.



tables in training and validation/test. For fine-tuning, we initialize the parameters with a pre-trained model, and further train all parameters with a task-specific objective. To demonstrate the efficiency of pre-training, we only fine-tune our model for 10 epochs unless otherwise stated.

6.2 Entity Linking

Entity linking is a fundamental task in table interpretation, which is defined as:

Definition 6.1. Given a table T and a knowledge base $\mathcal{KB}$, entity linking aims to link each potential mention in cells of T to its referent entity $e \in \mathcal{KB}$.

Entity linking is usually addressed in two steps: a candidate generation module first proposes a set of potential entities, and an entity disambiguation module then ranks and selects the entity that best matches the surface form and is most consistent with the table context. Following existing work [4, 17, 36], we focus on entity disambiguation and use an existing Wikidata Lookup service for candidate generation.

Baselines. We compare against the most recent methods for table entity linking T2K [36], Hybrid II [17], as well as the off-the-shelf Wikidata Lookup service. T2K uses an iterative matching approach that combines both schema and entity matching. Hybrid II [17] combines a lookup method with an entity embedding method. For Wikidata Lookup, we simply use the top-1 returned result as the prediction.

Fine-tuning TURL. Entity disambiguation is essentially matching a table cell with candidate entities. We treat each cell as a potential entity, and input its cell text (entity mention $\mathbf{e}^{\mathrm{m}}$ in Eqn. 2) as

well as table metadata to our Transformer encoder and obtain a contextualized representation $\mathbf{h}^{e}$ for each cell. To represent each candidate entity, we utilize the name and description as well as type information from a KB. The intuition is that when the candidate generation module proposes multiple entity candidates with similar names, we will utilize the description and type information to find the candidate that is most consistent with the table context. Specifically, for a KB entity e, given its name N and description D (both are a sequence of words) and types T, we get its representation $\mathbf{e}^{\text{kb}}$ as follows:

$$\mathbf{e}^{\text{kb}} = [\texttt{MEAN}_{w \in N}(\mathbf{w}), \texttt{MEAN}_{w \in D}(\mathbf{w}), \texttt{MEAN}_{t \in T}(\mathbf{t})]. \quad (8)$$

Here, $\mathbf{w}$ is the embedding for word w, which is shared with the embedding layer of pre-trained model. $\mathbf{t}$ is the embedding for entity type t to be learned during this fine-tuning phase. We then calculate a matching score between $\mathbf{e}^{\text{kb}}$ and $\mathbf{h}^{e}$ similarly as Eqn. 6. We do not use the entity embeddings pre-trained by our model here, as the goal is to link mentions to entities in a target KB, not necessarily those appear in our pre-training table corpus. The model is fine-tuned with a cross-entropy loss.

Task-specific Datasets. We use three datasets to compare different entity linking models: (1) We adopt the Wikipedia gold standards (WikiGS) dataset from [17], which contains 4,453,329 entity mentions extracted from 485,096 Wikipedia tables and links them to DBpedia [3]. (2) Since tables in WikiGS also come from Wikipedia, some of the tables may have already been seen during pre-training, despite their entity linking information is mainly used to pre-train entity embeddings (which are not used here). For a better comparison, we also create our own test set from the held-out test tables mentioned in Section 5.1, which contains 297,018 entity mentions from 4,964 tables. (3) To test our model on Web tables (i.e., those from websites other than Wikipedia), we also include the T2D dataset [26] which contains 26,124 entity mentions from 233 Web tables.[1] We use names and descriptions returned by Wikidata Lookup, and entity types from DBpedia.

The training set for fine-tuning TURL is based on our pre-training corpus, but with tables in the above WikiGS removed. We also remove duplicate entity mentions and mentions where Wikidata Lookup fails to return the ground truth entity in candidates, and finally obtain 1,264,217 entity mentions in 192,728 tables to fine-tune our model for the entity linking task.

Results. We set the maximum candidate size for Wikidata Lookup at 50 and also include the result of a Wikidata Lookup (Oracle), which considers an entity linking instance as correct if the ground-truth entity is in the candidate set. Due to lack of open-sourced implementations, we directly use the results of T2K and Hybrid II in [17]. We use F1, precision (P) and recall (R) measures for evaluation. False positive is the number of mentions where the model links to wrong entities, not including the cases where the model makes no prediction (e.g., Wikidata Lookup returns empty candidate set).

As shown in Table 4, our model gets the best F1 score and substantially improves precision on WikiGS and our own test set. The disambiguation accuracy on WikiGS is 89.62% (predict the correct entity if it is in the candidate set). A more advanced candidate generation module can help achieve better results in the future.

[1]We use the data released by [17] (https://doi.org/10.6084/m9.figshare.5229847).

Table 4: Model evaluation on entity linking task. All three datasets are evaluated with the same TURL + fine-tuning model.

Method	WikiGS			Our Test Set			T2D		
	F1	P	R	F1	P	R	F1	P	R
T2K [36]	34	70	22	-	-	-	82	**90**	76
Hybrid II [17]	64	69	**60**	-	-	-	**83**	85	**81**
Wikidata Lookup	57	67	49	62	62	60	80	86	75
TURL + fine-tuning	**67**	**79**	58	**68**	**71**	**66**	78	83	73
w/o entity desc.	60	70	52	60	63	58	-	-	-
w/o entity type	66	78	57	67	70	65	-	-	-
+ reweighting	-	-	-	-	-	-	82	88	77
WikiLookup (Oracle)	74	88	64	79	82	76	90	96	84

We also conduct an ablation study on our model by removing the description or type information of a candidate entity from Eqn. 8. From Table 4, we can see that entity description is very important for disambiguation, while entity type information only results in a minor improvement. This is perhaps due to the incompleteness of DBpedia, where a lot of entities have no types assigned or have missing types.

On the T2D dataset, all models perform much better than on the two Wikipedia datasets, mainly because of its smaller size and limited types of entities. The Wikidata Lookup baseline achieved high performance, and re-ranking using our model does not further improve. However, we adopt simple reweighting[2] to take into account the original result returned by Wikidata Lookup, which brings the F1 score to 0.82. This demonstrates the potential of using features such as entity popularity (used in Wikidata Lookup) and ensembling strong base models. Additionally, we conduct an error analysis on T2D comparing our model (TURL + fine-tuning + reweighting) with Wikidata Lookup. From Table 5, we can see that while in many cases, our model can infer the correct entity type based on the context and re-rank the candidate list accordingly, it makes mistakes when there are entities in the KB that look very similar to the mentions. To summarize, Table 4 and 5 show that there is room for further improvement of our model on entity linking, which we leave as future work.

6.3 Column Type Annotation

We define the task of column type annotation as follow:

Definition 6.2. Given a table T and a set of semantic types $\mathcal{L}$, column type annotation refers to the task of annotating a column in T with $l \in \mathcal{L}$ so that all entities in the column have type l. Note that a column can have multiple types.

Column type annotation is a crucial task for table understanding and is a fundamental step for many downstream tasks like data integration and knowledge discovery. Earlier work [30, 36, 49] on column type annotation often coupled the task with entity linking. First entities in a column are linked to a KB and then majority voting is employed on the types of the linked entities. More recently, [10, 11, 22] have studied column type annotation based on cell texts only. Here we adopt a similar setting, i.e., use the available information in a given table directly for column type annotation without performing entity linking first.

[2]We simply reweight the score of the top-1 prediction by our model with a factor of 0.8 and compare it with the top-1 prediction returned by Wikidata Lookup. The higher one is chosen as final prediction.

Table 5: Further analysis for entity linking on T2D corpus.

Mention	Page title	Header	Wikidata Lookup result	TURL + fine-tuning + reweighting result	Improve
philip	List of saints	Saint	Philip, male given name	Philip the Apostle, Christian saint and apostle	Yes
bank of nova scotia	The Global 2000 - Forbes.com	Company	Scotiabank, Canadian bank based in Toronto	Bank of Nova Scotia, bank building in Calgary	No
purple finch	The Sea Ranch Association List of Birds	Common Name	Haemorhous purpureus, species of bird	Purple Finch, print in the National Gallery of Art	No

Baselines. We compare our results with the state-of-the-art model Sherlock [22] for column type annotation. Sherlock uses 1588 features describing statistical properties, character distributions, word embeddings, and paragraph vectors of the cell values in a column. It was originally designed to predict a single type for a given column. We change its final layer to $|\mathcal{L}|$ Sigmoid activation functions, each with a binary cross-entropy loss, to fit our multi-label setting. We also evaluate our model using two datasets in [11], and include the HNN + P2Vec model as baseline. HNN + P2Vec employs a hybrid neural network to extract cell, row and column features, and combines it with property features retrieved from KB.

Fine-tuning TURL. To predict the type(s) for a column, we first extract the contextualized representation of the column $\mathbf{h}_c$ as follows:

$$\mathbf{h}_c = [\texttt{MEAN}\left(\mathbf{h}_i^{\text{t}}, \dots\right); \texttt{MEAN}\left(\mathbf{h}_j^{\text{e}}, \dots\right)]. \tag{9}$$

Here $\mathbf{h}_i^{\text{t}}$'s are representations of tokens in the column header, $\mathbf{h}_j^{\text{e}}$'s are representations of entity cells in the column. The probability of predicting type l is then given as,

$$P(l) = \text{Sigmoid}\left(\mathbf{h}_c W_l + b_l\right). \tag{10}$$

Same as with the baselines, we optimize the binary cross-entropy loss, y is the ground truth label for type l

$$loss = \sum y \log\left(P(l)\right) + (1-y)\log\left(1-P(l)\right) \tag{11}$$

Task-specific Datasets. We refer to Freebase [20] to obtain semantic types $\mathcal{L}$ because of its richness, diversity, and scale. We only keep those columns in our relational table corpus that have at least three linked entities to Freebase, and for each column, we use the common types of its entities as annotations. We further filter out types with less than 100 training instances and keep only the most representative types. In the end, we get a total number of 255 types, 628,254 columns from 397,098 tables for training, 13,025 (13,391) columns from 4,764 (4,844) tables for test (validation). We also test our model on two existing small-scale datasets, T2D-Te and Efthymiou (a subset of WikiGS annotated with types) from [11] and conduct two auxiliary experiments: (1) We first directly test our trained models and see how they generalize to existing datasets. We manually map 24 out of the 37 types used in [11] to our types, which results in 107 (of the original 133) columns in T2D-Te and 416 (of the original 614) columns in Efthymiou. (2) We follow the setting in [11] and use 70% of T2D as training data, which contains 250 columns.[3]

Results. For the main results on our test set, we use the validation set for early stopping in training the Sherlock model, which takes over 100 epochs. We evaluate model performance using F1, Precision (P) and Recall (R) measures. Results are shown in Table 6. Our model substantially outperforms the baseline, even when using the same input information (only entity mention vs Sherlock). Adding table metadata information and entity embedding learned during pre-training further boost the performance to 94.75 under F1. In addition, our model achieves such performance using only 10 epochs for fine-tuning, which demonstrates the efficiency of the pre-training/fine-tuning paradigm. More detailed results for several types are shown in Table 9, where we observe that all methods work well for coarse-grained types like `person`. However, fine-grained types like `actor` and `pro_athlete` are much more difficult to predict. Specifically, it is hard for a model to predict such types for a column only based on entity mentions in cells. On the other hand, using table metadata works much better than using entity mentions (e.g., 70.86 vs 58.86 for `actor`). This indicates the importance of table context information for predicting fine-grained column types.

Results of the auxiliary experiments are summarized in Table 7 and 8. The scores shown are accuracy, i.e., the ratio of correctly labeled columns, given each column is annotated with one ground

[3]We use the data released by [11] (https://github.com/alan-turing-institute/SemAIDA). The number of instances is slightly different from the original paper.

Table 6: Model evaluation on column type annotation task.

Method	F1	P	R
Sherlock (only entity mention) [22]	78.47	88.40	70.55
TURL + fine-tuning (only entity mention)	88.86	90.54	87.23
TURL + fine-tuning	**94.75**	**94.95**	**94.56**
w/o table metadata	93.77	94.80	92.76
w/o learned embedding	92.69	92.75	92.63
only table metadata	90.24	89.91	90.58
only learned embedding	93.33	94.72	91.97

Table 7: Accuracy on T2D-Te and Efthymiou, where scores for HNN + P2Vec are copied from [11] (trained with 70% of T2D and Efthymiou respectively and tested on the rest). We directly apply our models by type mapping without retraining.

Method	T2D-Te	Efthymiou
HNN + P2Vec (entity mention + KB) [11]	0.966	0.865
TURL + fine-tuning (only entity mention)	0.888	0.745
+ table metadata	0.860	0.904

Table 8: Accuracy on T2D-Te and Efthymiou. Here all models use T2D-Tr (70% of T2D) as training set, following the setting in [11].

Method	T2D-Te	Efthymiou
HNN + P2Vec (entity mention + KB) [11]	0.966	0.650
TURL + fine-tuning (only entity mention)	0.940	0.516
+ table metadata	0.962	0.746

Table 9: Further analysis on column type annotation: Model performance for 3 selected types. Results are F1 on validation set.

Method	person	pro_athlete	actor
Sherlock [22]	96.85	74.39	29.07
TURL + fine-tuning	**99.71**	**91.14**	**74.85**
only entity mention	98.44	87.11	58.86
only table metadata	98.26	88.80	70.86
only learned embedding	98.72	91.06	73.62

Table 10: Model evaluation on relation extraction task.

Method	F1	P	R
BERT-based	90.94	91.18	90.69
TURL + fine-tuning (only table metadata)	92.13	91.17	93.12
TURL + fine-tuning	**94.91**	**94.57**	**95.25**
w/o table metadata	93.85	93.78	93.91
w/o learned embedding	93.35	92.90	93.80

Table 11: Relation extraction results of an entity linking based system, under different agreement ratio thresholds.

Min Ag. Ratio	F1	P	R
0	68.73	60.33	**79.85**
0.4	82.10	94.65	72.50
0.5	77.68	98.33	64.20
0.7	63.10	99.37	46.23

truth label. For HNN + P2Vec, the scores are directly copied from the original paper [11]. Note that in Table 7, the numbers from our models are not directly comparable with HNN + P2Vec, due to mapping the types in the original datasets to ours as mentioned earlier. However, taking HNN + P2Vec trained on in-domain data as reference, we can see that without retraining, our models still obtain high accuracy on both Web table corpus (T2D-Te) and Wikipedia table corpus (Efthymiou). We also notice that adding table metadata slightly decreases the performance on T2D while increasing that on Efthymiou, which is possibly due to the distributional differences between Wikipedia tables and general web tables. From Table 8 we can see that when trained on the same T2D-Tr split, our model with both entity mention and table metadata still outperforms or is on par with the baseline. However, when using only entity mention, our model does not perform as well as the baseline, especially when generalizing to Efthymiou. This is because: (1) Our model is pre-trained with both table metadata and entity embedding. Removing both creates a big mismatch between pretraining and fine-tuning. (2) With only 250 training instances, it is easy for deep models to overfit. The better performance of models leveraging table metadata under both settings demonstrates the usefulness of context for table understanding.

6.4 Relation Extraction

Relation extraction is the task of mapping column pairs in a table to relations in a KB. A formal definition is given as follows.

Definition 6.3. Given a table T and a set of relations $\mathcal{R}$ in KB. For a subject-object column pair in T, we aim to annotate it with $r \in \mathcal{R}$ so that r holds between all entity pairs in the columns.

Most existing work [30, 36, 49] assumes that all relations between entities are known in KB and relations between columns can be easily inferred based on entity linking results. However, such methods rely on entity linking performance and suffer from KB incompleteness. Here we aim to conduct relation extraction without explicitly linking table cells to entities. *This is important as it allows the extraction of new knowledge from Web tables for tasks like knowledge base population.*

Baselines. We compare our model with a state-of-the-art text based relation extraction model [50] which utilizes a pretrained BERT model to encode the table information. For text based relation extraction, the task is to predict the relation between two entity

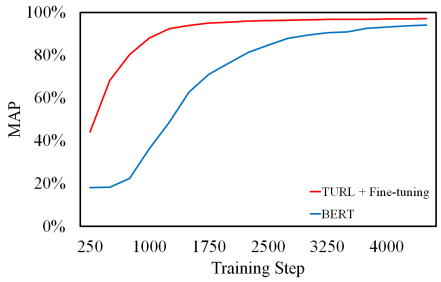


Figure 6: Comparison of fine-tuning our model and BERT for relation extraction: Our model converges much faster.

mentions in a sentence. Here we adapt the setting by treating the concatenated table metadata as a sentence, and the headers of the two columns as entity mentions. Although our setting is different from the entity linking based relation extraction systems in [30, 36, 49], here we implement a similar system using our entity linking model described in Section 6.2, and obtain relation annotations based on majority voting of linked entity pairs, i.e., predict a relation if it holds between a minimum portion of linked entity pairs in KB (i.e., the minimum agreement ratio is larger than a threshold).

Fine-tuning TURL. We use similar model architecture as column type annotation as follows.

$$P(r) = \text{Sigmoid}\left([\mathbf{h}_c; \mathbf{h}_{c'}]W_r + b_r\right). \tag{12}$$

Here $\mathbf{h}_c, \mathbf{h}_{c'}$ are aggregated representation for the two columns obtained same as Eqn. 9. We use binary cross-entropy loss for optimization.

Task-specific Datasets. We prepare datasets for relation extraction in a similar way as the previous column type annotation task, based on our pre-training table partitions. Specifically, we obtain relations $\mathcal{R}$ from Freebase. For each table in our corpus, we pair its subject column with each of its object columns, and annotate the column pair with relations shared by more than half of the entity pairs in the columns. We only keep relations that have more than 100 training instances. Finally, we obtain a total number of 121 relations, 62,954 column pairs from 52,943 tables for training, and 2072 (2,175) column pairs from 1467 (1,560) tables for test (validation).

Results. We fine-tune the BERT-based model for 25 epochs. We use F1, Precision (P) and Recall (R) measures for evaluation. Results are summarized in Table 10.

From Table 10 we can see that: (1) Both the BERT-based baseline and our model achieve good performance, with F1 scores larger than 0.9. (2) Our model outperforms the BERT-based baseline under all settings, even when using the same information (i.e., only table metadata vs BERT-based). Moreover, we plot the mean average precision (MAP) curve on our validation set during training in Figure 6. As one can see, our model converges much faster in comparison to the BERT-based baseline, demonstrating that our model learns a better initialization through pre-training.

As mentioned earlier, we also experiment with an entity linking based system. Results are summarized in Table 11. We can see that it can achieve high precision, but suffers from low recall: The upper-bound of recall is only 79.85%, achieved at an agreement ratio of 0 (i.e., taking all relations that exist between the linked entity pairs as positive). As seen from Table 10 and 11, our model also substantially outperforms the system based on a strong entity linker.

6.5 Row Population

Row population is the task of augmenting a given table with more rows or row elements. For relational tables, existing work has tackled this problem by retrieving entities to fill the subject column [46, 48]. A formal definition of the task is given below.

Definition 6.4. Given a partial table T, and an optional set of seed subject entities, row population aims to retrieve more entities to fill the subject column.

Baselines. We adopt models from [46] and [13] as baselines. [46] uses a generative probabilistic model which ranks candidate entities considering both table metadata and entity co-occurrence statistics. [13] further improves upon [46] by utilizing entity embeddings trained on the table corpus to estimate entity similarity. We use the same candidate generation module from [46] for all methods, which formulates a search query using either the table caption or seed entities and then retrieves tables via the BM25 retrieval algorithm. Subject entities in those retrieved tables will be candidates for row population.

Fine-tuning TURL. We adopt the same candidate generation module used by baselines. We then append the [MASK] token to the input, and use the hidden representation $\mathbf{h}^e$ of [MASK] to rank these candidates as shown in Table 3. We fine-tune our model with multi-label soft margin loss as shown below:

$$P(e) = \text{Sigmoid}\left(\texttt{LINEAR}(\mathbf{h}^e) \cdot \mathbf{e}^e\right),$$
$$loss = \sum_{e \in \mathcal{E}_C} y \log\left(P(e)\right) + (1-y)\log\left(1-P(e)\right). \quad (13)$$

Here $\mathcal{E}_C$ is the candidate entity set, and y is the ground truth label of whether e is a subject entity of the table.

Task-specific Datasets. Tables in our pre-training set with more than 3 subject entities are used for fine-tuning TURL and developing baseline models, while tables in our held-out set with more than 5 subject entities are used for evaluation. In total, we obtain 432,660 tables for fine-tuning with 10 subject entities on average, and 4,132 (4,205) tables for test (validation) with 16 (15) subject entities on average.

Results. The experiments are conducted under two settings: without any seed entity and with one seed entity. For experiments without the seed entity, we only use table caption for candidate generation. For entity ranking in EntiTables [46], we use the combination of caption and label likelihood when there is no seed entity, and only use entity similarity when seed entities are available. This strategy works best on our validation set. As shown in Table 12, our method outperforms all baselines. In particular, previous methods rely on entity similarity and are not applicable or have poor results when there is no seed entity available. Our method achieves a decent performance even without any seed entity, which demonstrates the effectiveness of TURL for generating contextualized representations based on both table metadata and content.

6.6 Cell Filling

We examine the utility of our model in filling other table cells, assuming the subject column is given. This is similar to the setting in [43, 47], which we formally define as follows.

Table 12: Model evaluation on row population task. Recall is the same for all methods because they share the same candidate generation module.

# seed	0		1	
Method	MAP	Recall	MAP	Recall
EntiTables [46]	17.90	63.30	42.31	78.13
Table2Vec [13]	-	63.30	20.86	78.13
TURL + fine-tuning	**40.92**	63.30	**48.31**	78.13

Table 13: Model evaluation on cell filling task.

Method	P @ 1	P @ 3	P @ 5	P @ 10
Exact	51.36	70.10	76.80	84.93
H2H	51.90	70.95	77.33	85.44
H2V	52.23	70.82	77.35	85.58
TURL	**54.80**	**76.58**	**83.66**	**90.98**

Table 14: Model evaluation on schema augmentation task.

Method	#seed column labels	
	0	1
kNN	80.16	**82.01**
TURL + fine-tuning	**81.94**	77.55

Definition 6.5. Given a partial table T with the subject column filled and an object column header, cell filling aims to predict the object entity for each subject entity.

Baselines. We adopt [47] as our base model. It has two main components, candidate value finding and value ranking. The same candidate value finding module is used for all methods: Given a subject entity e and object header h for the to-be-filled cells, we find all entities that appear in the same row with e in our pre-training table corpus , and only keep entities whose source header h' is related to h. Here we use the formula from [47] to measure the relevance of two headers $P(h'|h)$,

$$P(h'|h) = \frac{n(h',h)}{\sum_{h''} n(h'',h)}. \quad (14)$$

Here $n(h',h)$ is the number of table pairs in the table corpus that contain the same entity for a given subject entity in columns h' and h. The intuition is that if two tables contain the same object entity for a given subject entity e in columns with headings h_a and h_b, then h_a and h_b might refer to the same attribute. For value ranking, the key is to match the given header h with the source header h', we can then get the probability of the candidate entity e belongs to the cell $P(e|h)$ as follows:

$$P(e|h) = \text{MAX}\left(\text{sim}(h',h)\right). \quad (15)$$

Here h''s are the source headers associated with the candidate entity in the pre-training table corpus. $\text{sim}(h',h)$ is the similarity between h' and h. We develop three baseline methods for $\text{sim}(h',h)$: (1) **Exact**: predict the entity with exact matched header, (2) **H2H**: use the $P(h'|h)$ described above. (3) **H2V**: similar to [13], we train header embeddings with Word2Vec on the table corpus. We then measure the similarity between headers using cosine similarity.

Fine-tuning TURL. Since cell filling is very similar to the MER pre-training task, we do not fine-tune the model, and directly use [MASK] to select from candidate entities same as MER (Eqn. 6).

Task-specific Datasets. To evaluate different methods on this task, we use the held-out test tables in our pre-training phase and extract from them those subject-object column pairs that have at least

Table 15: Case study on schema augmentation. Here we show average precision (AP) for each example. Support Caption is the caption of the source table that kNN found to be most similar to the query table. Our model performs worse when there exist source tables that are very similar to the query table (e.g., comparing support caption vs query caption).

Method	Query Caption	Seed	Target	AP	Predicted	Support Caption
kNN	2010 santos fc season out	pos.	name, moving to	1.0	moving to, name, player, moving from, to	2007 santos fc season out
Ours				0.58	moving to, fee/notes, destination club, fee, loaned to	-
kNN	list of radio stations in metro manila am stations	name	format, covered location	1.0	format, covered location, company, call sign, owner	list of radio stations in metro manila fm stations
Ours				0.83	format, owner, covered location, city of license, call sign	-

three valid entity pairs. Finally we obtain 9,075 column pairs for evaluation.

Results. For candidate value finding, using all entities appearing in the same row with a given subject entity e achieves a recall of 62.51% with 165 candidates on average. After filtering with $P(h'|h) > 0$, the recall drops slightly to 61.45% and the average number of candidates reduces to 86. For value ranking, we only consider those test instances with the target object entity in the candidate set and evaluate them under Precision@K (or, P@K). Results are summarized in Table 13, from which we show: (1) Simple **Exact** match achieves decent performance, and using **H2H** or **H2V** only sightly improves the results. (2) Even though our model directly ranks the candidate entities without explicitly using their source table information, it outperforms other methods. This indicates that our model already encodes the factual knowledge in tables into entity embeddings through pre-training.

6.7 Schema Augmentation

Aside from completing the table content, another direction of table augmentation focuses on augmenting the table schema, i.e., discovering new column headers to extend a table with more columns [6, 13, 43, 46]. Following [13, 43, 46], we formally define the task below.

Definition 6.6. Given a partial table T, which has a caption and zero or a few seed headers, and a header vocabulary $\mathcal{H}$, schema augmentation aims to recommend a ranked list of headers $h \in \mathcal{H}$ to add to T.

Baselines. We adopt the method in [46] which searches our pre-training table corpus for related tables, and use headers in those related tables for augmentation. More specifically, we encode the given table caption as a tf-idf vector and then use the K-nearest neighbors algorithm (kNN) [2] with cosine similarity to find the top-10 most related tables. We rank headers from those tables by aggregating the cosine similarities for tables they belong to. When seed headers are available, we re-weight the tables by the overlap of their schemas with seed headers same as [46].

Fine-tuning TURL. We concatenate the table caption, seed headers and a [MASK] token as input to our model. The output for [MASK] is then used to predict the headers in a given header vocabulary $\mathcal{H}$. We fine-tune our model use binary cross-entropy loss.

Task-specific Datasets. We collect $\mathcal{H}$ from the pre-training table corpus. We normalize the headers using simple rules, only keep those that appear in at least 10 different tables, and finally obtain 5652 unique headers, with 316,858 training tables and 4,646 (4,708) test (validation) tables.

Results. We fine-tune our model for 50 epochs for this task, based on the performance on the validation set. We use mean average precision (MAP) for evaluation.

From Table 14, we observe that both kNN baseline and our model achieve good performance. Our model works better when no seed header is available, but does not perform as well when there is one seed header. We then conduct a further analysis in Table 15 using a few examples: One major reason why kNN works well is that there exist tables in the pre-training table corpus that are very similar to the query table and have almost the same table schema. On the other hand, our model oftentimes suggests plausible, semantically related headers, but misses the ground-truth headers.

7 CONCLUSION

This paper presents a novel pre-training/fine-tuning framework (TURL) for relational table understanding. It consists of a structure-aware Transformer encoder to model the row-column structure as well as a new Masked Entity Recovery objective to capture the semantics and knowledge in relational Web tables during pre-training. On our compiled benchmark, we show that TURL can be applied to a wide range of tasks with minimal fine-tuning and achieves superior performance in most scenarios. Interesting future work includes: (1) Focusing on other types of knowledge such as numerical attributes in relational Web tables, in addition to entity relations. (2) Incorporating the rich information contained in an external KB into pre-training.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their helpful comments. Authors at the Ohio State University were sponsored in part by Google Faculty Award, the Army Research Office under cooperative agreements W911NF-17-1-0412, NSF Grant IIS1815674, NSF CAREER #1942980, Fujitsu gift grant, and Ohio Supercomputer Center [9]. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

REFERENCES

[1] Ahmad Ahmadov, Maik Thiele, Julian Eberius, Wolfgang Lehner, and Robert Wrembel. 2015. Towards a hybrid imputation approach using web tables. In *2015 IEEE/ACM 2nd International Symposium on Big Data Computing (BDC)*. IEEE, 21–30.

[2] Naomi S. Altman. 1992. An Introduction to Kernel and Nearest Neighbor Nonparametric Regression.

[3] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. 2007. DBpedia: A Nucleus for a Web of Open Data. In *ISWC/ASWC*.

[4] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. 2015. TabEL: Entity Linking in Web Tables. In *Proceedings of the 14th International Conference on The Semantic Web-ISWC 2015-Volume 9366*. 425–441.

[5] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *NIPS*.

[6] Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. 2008. Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment* 1, 1 (2008), 538–549.

[7] Michael J. Cafarella, Alon Y. Halevy, Yang Zhang, Daisy Zhe Wang, and Eugene Wu. 2008. Uncovering the Relational Web. In *11th International Workshop on the Web and Databases, WebDB 2008, Vancouver, BC, Canada, June 13, 2008*.

[8] Matteo Cannaviccio, Lorenzo Ariemma, Denilson Barbosa, and Paolo Merialdo. 2018. Leveraging wikipedia table schemas for knowledge graph augmentation. In *Proceedings of the 21st International Workshop on the Web and Databases*. 1–6.

[9] Ohio Supercomputer Center. 1987. Ohio Supercomputer Center. http://osc.edu/ark:/19495/f5s1ph73

[10] Jiaoyan Chen, Ernesto Jiménez-Ruiz, Ian Horrocks, and Charles A. Sutton. 2018. ColNet: Embedding the Semantics of Web Tables for Column Type Prediction. In *AAAI*.

[11] Jiaoyan Chen, Ernesto Jiménez-Ruiz, Ian Horrocks, and Charles A. Sutton. 2019. Learning Semantic Annotations for Tabular Data. In *IJCAI*.

[12] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. 2012. Finding Related Tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 817âĂŞ828.

[13] Lei Min Deng, Shuo Zhang, and Krisztian Balog. 2019. Table2Vec: Neural Word and Entity Embeddings for Table Population and Retrieval. In *SIGIR'19*.

[14] Xiang Deng and Huan Sun. 2019. Leveraging 2-hop Distant Supervision from Table Entity Pairs for Relation Extraction. *arXiv preprint arXiv:1909.06007* (2019).

[15] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. TURL: Table Understanding through Representation Learning. https://arxiv.org/abs/2006.14806. arXiv:2006.14806 [cs.IR]

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 4171–4186.

[17] 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 *International Semantic Web Conference*.

[18] Raul Castro Fernandez and Samuel Madden. 2019. Termite: a system for tunneling through heterogeneous data. In *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*. 1–8.

[19] Xavier Glorot, Antoine Bordes, Jason Weston, and Yoshua Bengio. 2013. A semantic matching energy function for learning with multi-relational data. *Machine Learning* 94 (2013), 233–259.

[20] Google. 2015. Freebase Data Dumps. https://developers.google.com/freebase/data.

[21] Jonathan Herzig, P. Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. TAPAS: Weakly Supervised Table Parsing via Pre-training. In *ACL*.

[22] Madelon Hulsebos, Kevin Zeng Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, cCaugatay Demiralp, and C'esar A. Hidalgo. 2019. Sherlock: A Deep Learning Approach to Semantic Data Type Detection. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2019).

[23] Xiaoqi Jiao, Y. Yin, Lifeng Shang, Xin Jiang, Xusong Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. TinyBERT: Distilling BERT for Natural Language Understanding. *ArXiv* abs/1909.10351 (2019).

[24] Ernesto Jiménez-Ruiz, Oktie Hassanzadeh, Vasilis Efthymiou, Jiaoyan Chen, and Kavitha Srinivas. 2020. SemTab 2019: Resources to Benchmark Tabular Data to Knowledge Graph Matching Systems. In *European Semantic Web Conference*. Springer, 514–530.

[25] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2015).

[26] 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 Companion on World Wide Web*. 75–76.

[27] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *arXiv preprint arXiv:2004.00584* (2020).

[28] LimayeGirija, SarawagiSunita, and ChakrabartiSoumen. 2010. Annotating and searching web tables using entities, types and relationships. In *VLDB 2010*.

[29] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[30] Varish Mulwad, Timothy W. Finin, Zareen Syed, and Anupam Joshi. 2010. Using Linked Data to Interpret Tables. In *COLD*.

[31] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global Vectors for Word Representation. In *EMNLP*.

[32] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *NAACL-HLT*.

[33] Matthew E Peters, Mark Neumann, Robert Logan, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A Smith. 2019. Knowledge Enhanced Contextual Word Representations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 43–54.

[34] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. (2019).

[35] Sebastian Riedel, Limin Yao, Andrew McCallum, and Benjamin M. Marlin. 2013. Relation Extraction with Matrix Factorization and Universal Schemas. In *HLT-NAACL*.

[36] Dominique Ritze, Oliver Lehmberg, and Christian Bizer. 2015. Matching HTML Tables to DBpedia. In *WIMS '15*.

[37] Yoones A Sekhavat, Francesco Di Paolo, Denilson Barbosa, and Paolo Merialdo. [n.d.]. Knowledge Base Augmentation using Tabular Data.

[38] Qi shan Wang, Zhendong Mao, Biwu Wang, and Li Guo. 2017. Knowledge Graph Embedding: A Survey of Approaches and Applications. *IEEE Transactions on Knowledge and Data Engineering* 29 (2017), 2724–2743.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[40] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. 353–355.

[41] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zhigang Chen. 2014. Knowledge Graph Embedding by Translating on Hyperplanes. In *AAAI*.

[42] Jason Weston, Antoine Bordes, Oksana Yakhnenko, and Nicolas Usunier. 2013. Connecting Language and Knowledge Bases with Embedding Models for Relation Extraction. *ArXiv* abs/1307.7973 (2013).

[43] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. 2012. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 97–108.

[44] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In *NeurIPS*.

[45] Pengcheng Yin, Graham Neubig, Wen tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In *ACL*.

[46] Shuo Zhang and Krisztian Balog. 2017. Entitables: Smart assistance for entity-focused tables. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 255–264.

[47] Shuo Zhang and Krisztian Balog. 2019. Auto-completion for data cells in relational tables. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 761–770.

[48] Shuo Zhang and Krisztian Balog. 2020. Web Table Extraction, Retrieval, and Augmentation: A Survey. *ACM Transactions on Intelligent Systems and Technology (TIST)* 11 (2020), 1 – 35.

[49] Ziqi Zhang. 2017. Effective and efficient Semantic Table Interpretation using TableMiner+. *Semantic Web* 8 (2017), 921–957.

[50] Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. 2019. ERNIE: Enhanced Language Representation with Informative Entities. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 1441–1451.