Capturing Row and Column Semantics in Transformer Based Question Answering over Tables

Michael Glass[1], Mustafa Canim[1], Alfio Gliozzo[1],
Saneem Chemmengath[1], Vishwajeet Kumar[1], Rishav Chakravarti[1],
Avirup Sil[1], Feifei Pan[2], Samarth Bharadwaj[1], Nicolas Rodolfo Fauceglia[1]
[1] IBM Research AI [2] Rensselaer Polytechnic Institute
mrglass@us.ibm.com, mustafa@us.ibm.com, gliozzo@us.ibm.com,
saneem.cg@in.ibm.com, vishk024@in.ibm.com, rchakravarti@us.ibm.com,
avi@us.ibm.com, panf2@rpi.edu, samarth.b@in.ibm.com, nicolas.fauceglia@ibm.com

Abstract

Transformer based architectures are recently used for the task of answering questions over tables. In order to improve the accuracy on this task, specialized pre-training techniques have been developed and applied on millions of open-domain web tables. In this paper, we propose two novel approaches demonstrating that one can achieve superior performance on table QA task without even using any of these specialized pre-training techniques. The first model, called *RCI interaction*, leverages a transformer based architecture that independently classifies rows and columns to identify relevant cells. While this model yields extremely high accuracy at finding cell values on recent benchmarks, a second model we propose, called *RCI representation*, provides a significant efficiency advantage for online QA systems over tables by materializing embeddings for existing tables. Experiments on recent benchmarks prove that the proposed methods can effectively locate cell values on tables (up to ∼98% Hit@1 accuracy on WikiSQL lookup questions). Also, the interaction model outperforms the state-of-the-art transformer based approaches, pre-trained on very large table corpora (TaPas and TaBert), achieving ∼3.4% and ∼18.86% additional precision improvement on the standard WikiSQL benchmark[1].

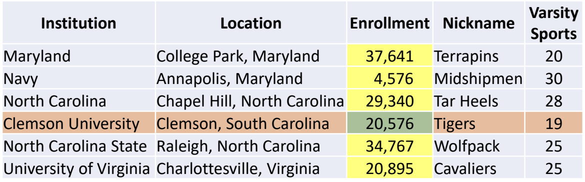
1 Introduction

Tabular data format is a commonly used layout in domain specific enterprise documents as well as open domain webpages to store structured information in a compact form (Pasupat and Liang, 2015; Canim et al., 2019). In order to make use of these resources, many techniques have been proposed for the retrieval of tables (Cafarella et al., 2008; Zhang and Balog, 2018; Venetis et al., 2011; Shraga et al., 2020; Sun et al., 2016). Given a large corpus of documents, the goal in these studies is to retrieve top-k relevant tables based on given keyword(s). The user is then expected to skim through these tables and locate the relevant cell values which is a tedious and time consuming task. More recently, popular search engines made significant improvement in understanding natural language questions and finding the answers within passages, owing to the developments in transformer based machine reading comprehension (MRC) systems (Rajpurkar et al., 2016, 2018; Kwiatkowski et al., 2019; Pan et al., 2019; Alberti et al., 2019a). One natural extension of these systems is to answer questions over tables. These questions are broadly classified into two types: *Lookup* and *Aggregation*. *Lookup* questions require returning exact strings from tables such as cell values whereas *Aggregation* questions are executed by performing an arithmetic operation on a subset of the column cells, such as *Min(), Max(), Average() and Count()*. For look-up questions, the users can verify if the returned cell values from the table(s) are correct, while this is not applicable for *Aggregation* questions because a scalar value is returned as an answer. Our primary focus in this paper is on *Lookup* questions since the answers are verifiable by users although our proposed techniques outperform the state-of-the-art (SOTA) approaches on both question types.

In this paper, we propose a new approach to table QA that independently predicts the probability of containing the answer to a question in each row and column of a table. By taking the **R**ow and **C**olumn **I**ntersection (RCI) of these probabilistic predictions, RCI gives a probability for each cell of the table. These probabilities are either used to answer questions directly or highlight the relevant regions of tables as a heatmap, helping users to easily locate the answers over tables (See Figure 1 for a question answered with the help of a heatmap). We developed two models for RCI, called *RCI interaction* and *RCI representation*.

[1] The source code and the models we built are available at https://github.com/IBM/row-column-intersection.

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



Institution	Location	Enrollment	Nickname	Varsity Sports
Maryland	College Park, Maryland	37,641	Terrapins	20
Navy	Annapolis, Maryland	4,576	Midshipmen	30
North Carolina	Chapel Hill, North Carolina	29,340	Tar Heels	28
Clemson University	Clemson, South Carolina	20,576	Tigers	19
North Carolina State	Raleigh, North Carolina	34,767	Wolfpack	25
University of Virginia	Charlottesville, Virginia	20,895	Cavaliers	25

Figure 1: Answering a question "What is the Clemson Tiger's enrollment?" over a table with a heatmap

In order to evaluate these approaches, we also propose a weakly supervised MRC system as a strong baseline to identify / "read" relevant cells of a table. In this baseline approach, we convert tables into passages and extract a relevant span of text within these passages.

The interaction model is designed to provide very high accuracy on finding cell values over tables for a given natural language question. We demonstrate that without even using any specialized pre-trained models, we can achieve up-to ~98% Hit@1 accuracy on finding cell values of tables for lookup questions from the WikiSQL benchmark. Also, the interaction model outperforms the state-of-the-art transformer based approaches, TAPAS (Herzig et al., 2020) and TABERT (Yin et al., 2020), achieving ~3.4% and ~18.86% additional precision improvement on the standard WikiSQL benchmark, containing both lookup and aggregation questions.

While the interaction model yields very high accuracy on the benchmarks, the representation model has the advantage of pre-computing the embeddings for all tables in a corpus and storing them for online query processing. Once a user query is received, the most relevant tables can be retrieved from a table retrieval system and relevant cell values can be highlighted using the existing embeddings of the tables, resulting in less computation per received user query, as opposed to running tables over expensive transformer architecture for every received query.

The specific contributions of this paper are as follows:

- **An MRC based strong baseline for table QA task:** We investigate a transfer learning approach by utilizing a fully supervised reading comprehension system built on top of a large pre-trained language model. Specifically, it is first fine-tuned on SQuAD then on Natural Questions and lastly trained on the table datasets. The final model is used to identify relevant cells of a table for a given natural language question.

- **A transformer based interaction model for the table QA task:** We propose a model for table QA task that concatenates a textual representation of each row (or column) to the text of the question and classifies the sequence pair as positive (the row/column contains the answer) or negative (the row/column does not contain the answer). The proposed approach yields very high accuracy on our benchmarks, outperforming the SOTA models.

- **A transformer based representation model for the table QA task:** We propose a representation model that builds vector representations of the question and each row (or column) to compare the resulting vectors to determine if the row (or column) contains the answer. The proposed approach is preferred for efficiency purposes on online table retrieval systems since it enables materializing embeddings for existing tables and re-using them during online question answering over multiple tables.

In the following sections, we first review the prior work on QA systems over tables as well as table search from large corpora in Section 2. We then describe a weakly supervised machine reading comprehension (MRC) system as a baseline that is capable of answering questions over tables in Section 3. In Section 4, we introduce two models that decompose TableQA as the intersection between rows and columns of a table using a transformer architecture. Experimental results are reported and discussed in Section 5 and finally Section 6 concludes the paper and discusses the future work.

2 Related Work

QA from text: There is plenty of work on QA from plain text (Brill et al., 2002; Lin, 2007; Paşca, 2003; Kwiatkowski et al., 2019; Pan et al., 2019). Typical strategies rely on token overlap between the question and passage text either based on a bag of word statistics or contextualized language model representations. In either case, tabular structure is not leveraged to capture semantic relationships between rows and columns. As we show in Section 5, these strategies are insufficient for answering questions over tables with high precision.

QA over tables: Our work mostly relates to the previous research on QA over tables (Pasupat and Liang, 2015; Sun et al., 2016; Dasigi et al., 2019). They center around answering factoid questions and return the exact cell of a table that answers the query. We briefly describe here how these works are different. Pasupat and Liang (2015) assume access to the 'gold' table that contains the answer to the input question. They build a semantic parser that parses the query to a logical form. They likewise convert the table into a knowledge-graph and execute the logical form on it to get the answer. A more advanced semantic parsing based methodology has been recently proposed by Dasigi et al. (2019). This system is pre-trained on WikiTablesQuestions (Pasupat and Liang, 2015). The proposed approach leverages an LSTM encoder-decoder model where tables are first converted to a knowledge-graph and word tokens in the questions are linked to table entities (columns and cells). The questions and linked table entities are then encoded into representation vectors which are decoded to executable λ-DCS logical forms. This logical forms are executed over a knowledge graph to get answer predictions. Our approach is different, since we do not convert natural language questions into logical forms and execute them on tables. Instead, we leverage transformer architectures pre-trained on large corpora and further trained on finding cell values on tables. In Section 5, we show that we achieve significant improvement over this approach without using any semantic parser technique.

Sun et al. (2016) focus on the table retrieval problem over table corpora by leveraging the content of cell values and headers. For a given query, they extract answers from millions of tables in the provided corpus. They construct a unified chain representation of both the input question and the table cells and then find the table cell chain that best matches the question chain. As opposed to this work, we primarily focus on answering questions over a single table rather than the retrieval of top-k tables from a corpus.

More recently, transformer based pre-training approaches have been introduced in TaBERT (Yin et al., 2020) and TaPas (Herzig et al., 2020) to improve accuracy for table QA. TaBERT has been pre-trained on 26 million tables and NL sentences extracted from Wikipedia and WDC WebTable Corpus (Yin et al., 2020). The model can be plugged into a neural semantic parser as an encoder to provide contextual embeddings for tables. Herzig et al. on the other hand, claim that semantic parsers incur an extra overhead of computing intermediate logical representations which can be avoided by leveraging fine-tuned models to answer questions over tables. The model in TaPas has been pre-trained on about 6 million tables extracted from Wikipedia content. Our work is different from both TaPas and TaBERT. First and foremost, our focus in this paper is not on pre-training a new model for table QA, but rather on leveraging the existing language models to find the connection between a question and table columns/rows with very high accuracy. Second, our goal is to provide a heatmap over tables on an end-to-end table retrieval system to help users to quickly identify the regions of tables where the answers would most likely appear. Because the transformer architectures are quite expensive to query, the representation model we propose radically reduces the computational overhead during online query processing.

Table search over the web: Another active research area in NLP is searching over web tables. There are numerous search algorithms that have been explored such as keyword search (Cafarella et al., 2008; Zhang and Balog, 2018; Venetis et al., 2011; Shraga et al., 2020), retrieve similar tables (Das Sarma et al., 2012), retrieve tables based on column names (Pimplikar and Sarawagi, 2012) and adding new columns to existing entity lists (Yakout et al., 2012; Zhang and Chakrabarti, 2013). This thread of work focuses on retrieval of top-k tables with high precision from large corpora, rather than finding relevant rows and columns within tables.

3 MRC Model

We provide a brief description of our underlying Machine Reading Comprehension (MRC) model architecture, which we use as a strong baseline. The architecture is inspired by (Alberti et al., 2019b; Pan et al., 2019; Glass et al., 2020) and direct interested readers to their papers for more details. Our MRC model follows the approach introduced by (Devlin et al., 2019) of starting with a pre-trained transformer based language model (LM) and then fine-tuning MRC specific feed-forward layers on both general question answering datasets (SQuAD 2.0 and NQ) as well as the table specific question answers associated with the datasets in Section 5.

We use ALBERT (Lan et al., 2020) as the underlying LM similar to models which achieve SOTA on the SQuAD 2.0 leaderboard (Zhang et al., 2020b,a) at the time of writing. More specifically, we show results starting from the weights and dimensions of the *base v2* version (25M parameters) of the LM shared by (Lan et al., 2020). We also experiment with the *xxlarge v2* version (235M parameters) as well. The input to the model is a token sequence ($\mathbf{X}$) consisting of a question, passage, and special markers (a $[CLS]$ token for answerability classification and $[SEP]$ tokens to dileneate between the query and passage). The input token sequence is passed through a deep Transformer (Vaswani et al., 2017) network to output a sequence of contextualized token representations $\mathbf{H}$.

MRC then adds two dense layers followed by a *softmax*:

$$\boldsymbol{\alpha}_b = softmax(\mathbf{W}_1\mathbf{H}),$$
$$\boldsymbol{\alpha}_e = softmax(\mathbf{W}_2\mathbf{H}),$$

where $\mathbf{W}_1$, $\mathbf{W}_2 \in \mathbb{R}^{1\times D_e}$. D_e denotes the dimensionality of the embeddings (768 for *base v2*). $\boldsymbol{\alpha}_b^t$ and $\boldsymbol{\alpha}_e^t$ denote the probability of the t^{th} token in the sequence being the answer beginning and end, respectively.

The model is trained using binary cross-entropy loss at each token position based on whether or not the annotated correct answer begins or ends at the t^{th} token. Unanswerable questions have their begin and end offsets set to the $[CLS]$ token position.

At prediction time, a score is calculated for each possible span by summing the $\boldsymbol{\alpha}_b^{t_j}$ and $\boldsymbol{\alpha}_e^{t_i}$ at each possible i and j combination to identify the max scoring answer span. The sum of the $\boldsymbol{\alpha}_b^{[CLS]}$ and $\boldsymbol{\alpha}_e^{[CLS]}$ is then subtracted from this max scoring answer span to produce a final score that can be used for thresholding (i.e., deciding whether to predict an answer or refrain from answering a question). A few modifications are made in line with (Alberti et al., 2019b) to use MRC for the NQ dataset which introduces additional answer types $[short, long, yes, no, null]$. Refer to the appendix for these details.

We fine-tune the model with the SQuAD 2.0 dataset and then the NQ dataset in line with (Pan et al., 2019; Glass et al., 2020), to produce a generic RC model comparable to the current SOTA. We then train for an additional epoch on the subset of NQ which consists of short answer questions that need to be answered by lookup inside an HTML table. This is about 5% of the total NQ data ($\sim 15,500$ question-answer pairs). Note that in these cases, the input "passage" text consists of textual representation of tables (i.e., we introduce tabs between columns and new line characters between rows); so it is devoid of true row and column structure. This pre-training and task adaptation strategy is inline with prior art (Gururangan et al., 2020) in adapting transformers. Simpler pre-training strategies (e.g. relying only on SQuAD 2.0 or skipping the table specific epoch of training) were tried and found to provide similar, but generally worse, performance. So those are excluded from Section 5 for brevity.

Finally, we fine-tune (i.e., train for an additional epoch) on the training examples (table-question pairs) associated with the appropriate evaluation data sets described in Section 5. During this step we do not have access to exact span offsets in the ground truth annotations and, instead, use weak supervision by matching the first occurrence of the answer text within the textual representation of the table[2].

4 RCI Model Architecture

The Row-Column Intersection model (RCI) is motivated by the idea of decomposing lookup Table QA into two operations: the column selection and the row selection. Combining the predicted answer probability of each row and the probability of each column gives a score for all cells in the table. The highest scoring cell may then be returned as an answer, or highlighting may be applied to the table to aid a user in locating the relevant information. Unlike the pointer network of an adapted Machine Reading Comprehension system (described in Section 3), the RCI model always gives a ranked list of cells rather than answer spans that may cross cell boundaries.

We observe that the process of identifying the correct column is often about matching the column header and the type of values in the column to the expected answer type of the question. For example in Table 1, the question has a lexical answer type of ‘party’ and the column header for the correct column is ‘Party’ and contains values that are political parties.

Identifying the correct row is often more difficult. In the example given in Table 1, it is sufficient to match either of the names in the question to the

[2] We provide the hyperparameters for the training process in the appendix.

value in the ‘Name’ column of the row. Note that with weak supervision (Min et al., 2019) we do not know the correct row, so all occurrences of ‘Pro-Administration’ are considered correct.

Name	Took office	Left office	Party	Notes / Events
Benjamin Contee	1789	1791	Anti-Administration	
William Pinkney	1791	1791	Pro-Administration	resigned
John Francis Mercer	1792	1793	Anti-Administration	
Uriah Forrest	1793	1794	Pro-Administration	resigned
Benjamin Edwards	1795	1795	Pro-Administration	
⋮				

What party was William Pinkney and Uriah Forrest a part of?

Answer: Pro-Administration

Table 1: Example TableQA over Wikipedia Table

Both the Row and Column models of RCI are sequence-pair classifiers. The question is one sequence and the text sequence representation of the row or column is the second sequence. We consider two approaches to the sequence-pair classification task in RCI: Interaction and Representation. Interaction models use the self attention of a transformer over the concatenated two sequences. This is the standard approach to sequence-pair classification tasks, *e.g.* textual entailment (Devlin et al., 2019) (Wang et al., 2018), in transformer based systems.

Representation models independently project each sequence of the sequence-pair to a vector, then compare those vectors. Representation models are motivated by the need to improve efficiency for a practical system. Considering the column classifier, the interaction model requires running a transformer over each question plus column sequence. In contrast, the representation model can pre-process the collection of tables, producing a vector representation of each column for each table, independent of any query. Then, at query time, the query is projected to a vector which is then combined with the vector for each column and classified with a single-layer network. On the WikiTableQuestions-Lookup dev set, we see the column model’s time drop from 40 seconds to 0.8 seconds on a K80 GPU when ten queries are batch processed at once.

Let a table with m rows and n columns be defined as a header, $H = [h_1, h_2, ..., h_n]$ and cell values $V = [v_{i,j}], 1 \leq i \leq m, 1 \leq j \leq n$. A TableQA instance consists of a table, a question and a ground truth set of cell indices, $T \subseteq I \times J, I = 1, 2, ..., m, J = 1, 2, ..., n$. In principle, these ground truth cell positions could be annotated with the correct occurrences of the correct values. However, this form of supervision may be too difficult to obtain. We use *weak supervision*: the ground truth cell indices are found by matching the ground truth answer strings in the table. To train the row and column classifier we find ground truth row and column indices:

$$T_r = \{i \mid \exists j : (i, j) \in T\}$$
$$T_c = \{j \mid \exists i : (i, j) \in T\}$$

Although it is possible to naïvely construct a sequence representation of columns and rows by simply space separating the contents of each row or column, better performance can be achieved by incorporating the table structure in the sequence representation. We focus on tables with a single header for columns, but this method could also be applied to tables with a hierarchical header, by first flattening the header.

The row (S^r_i) and column (S^c_j) sequence representations are formatted as:

$$S^r_i = \bigoplus_{j=1}^{n} \zeta_h(h_j) \oplus \ \zeta_v(v_{i,j})$$

$$S^c_j = \zeta_h(h_j) \oplus \bigoplus_{i=1}^{m} \zeta_v(v_{i,j})$$

Where $\oplus$ indicates concatenation and the functions ζ_h and ζ_v delimit the header and cell value contents. For ζ_h we append a colon token (‘:’) to the header string, and for ζ_v we append a pipe token (‘|’) to the cell value string. The particular tokens used in the delimiting functions are not important. Any distinctive tokens can serve since the transformer will learn an appropriate embedding to represent their role as header and cell value delimiters.

Considering again the example in Table 1, the first row would be represented as:

Name : Benjamin Contee | Took office : 1789 |
Left office : 1791 | Party : Anti-Administration |
Notes / Events : |

While the second column would have a sequence representation of:

Took office : 1789 | 1791 | 1792 | 1793 | 1795 |

Both the interaction and the representation models use the sequence representation described above. In the case of the interaction model this sequence is then appended to the question with standard $[CLS]$ and $[SEP]$ tokens to delimit the two sequences. This sequence pair is then input to a transformer encoder, ALBERT. The final hidden state for the $[CLS]$ token is used in a linear layer followed by a softmax to classify the column as either containing the answer or not.

In the representation model shown in Figure 2 the representations of the question (r_q) and the jth column sequence (r_c) are first computed independently. The representations are taken from the vector that the transformer model produces for the $[CLS]$ input token. These vectors are then concatenated (indicated as :) with their element-wise product (indicated as $\otimes$) and the element-wise square of their differences. The probability that this column is the target for the question is then given by a softmax over a linear layer.

$$\begin{aligned} \mathbf{r}_\delta &= \mathbf{r_q} - \mathbf{r_c} \\ \mathbf{v_{qc}} &= \mathbf{r_q} : \mathbf{r_c} : \mathbf{r_q} \otimes \mathbf{r_c} : \mathbf{r}_\delta \otimes \mathbf{r}_\delta \\ p(j \in T_c) &= softmax(\mathbf{W}\mathbf{v_{qc}} + \mathbf{b})_0 \end{aligned}$$

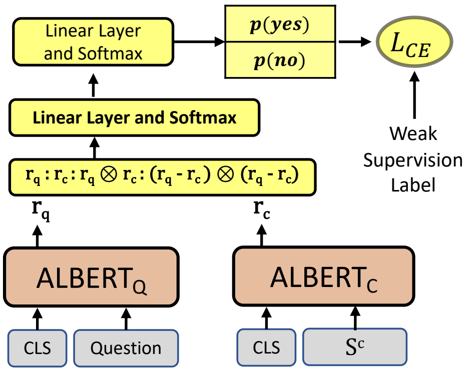


Figure 2: RCI Representation Model

Extension to aggregation questions: Although our focus is on lookup questions, the RCI model can be extended to aggregation questions with the addition of a question classifier. Another transformer is trained to classify the sequence-pair of the question and the table header into one of six categories: lookup, max, min, count, sum and average. The table header is relevant because a question such as "How many wins do the Cubs have?" can be lookup, count or sum depending on the structure of the table.

Taking a threshold on the cell level confidences of the RCI model and aggregating by the predicted question type produces the final answer, either a list of cells for lookup questions or a single number for aggregation questions.

This approach requires *full supervision*, we must know the cells to be aggregated to train the RCI row and column classifiers as well as the type of aggregation to train the question classifier. This type of supervision is available in the WikiSQL dataset, but not in WikiTableQuestions.

5 Evaluation

To evaluate these three approaches, we adapt three standard TableQA datasets: WikiSQL (Zhong et al., 2017), WikiTableQuestions (Pasupat and Liang, 2015) and TabMCQ (Jauhar et al., 2016). WikiSQL and WikiTableQuestions include both lookup questions as well as aggregation questions. As mentioned in Section 1, our primary focus in this paper is on *lookup* questions that require selection and projection operations over tables (i.e., identifying the row and column of a table with very high precision for a given natural language question). We are releasing the processing and evaluation code for the datasets to support reproducibility[3]. Table 2 gives a summary of these datasets.

In WikiSQL, the ground truth SQL query is provided for each question, so questions involving an aggregation operation can be automatically excluded. The lookup questions are 72% of the WikiSQL benchmark. WikiSQL has some questions ($< 3\%$) with multiple answers. We treat these as a list of relevant items and use information retrieval metrics to measure the quality of a predicted ranked list of cells.

TabMCQ is a multiple-choice, lookup TableQA dataset over general science tables. We discard the multiple-choice setting and treat it as a standard open-ended QA task. However, some TabMCQ tables are very large. Of the 68 tables, 17 have more than 50 rows, with two tables containing over a thousand rows. We down-sample the rows that are not relevant for a given question, limiting the largest table size to 50 rows. Unlike the other two datasets, these tables are not Wikipedia tables and have an unusual format. A sample TabMCQ table is provided in the appendix.

WikiTableQuestions does not provide a definitive indication for what questions are lookup questions. To identify these questions we first filter questions with words indicating an aggregation, such as 'average', 'min', 'max', etc. These questions were further filtered manually to get the WikiTableQuestions-Lookup set.

In order to evaluate our proposed approaches on these datasets we built three different systems

[3] https://github.com/IBM/row-column-intersection

Dataset	Train	Dev	Test
WikiSQL	40606	6017	11324
TabMCQ	5453	1819	1820
WikiTableQuestions	851	124	241

Table 2: Lookup TableQA Dataset Sizes

and also used three existing models: IS-SP, provided by (Dasigi et al., 2019), TaBERT (Yin et al., 2020) and TaPas (Herzig et al., 2020). IS-SP is a semantic parsing based model trained on WikiTablesQuestions (Pasupat and Liang, 2015) dataset (See Section 2 for the details of this work). For building their model we used the code provided in (Gardner et al., 2020). For TaBERT we trained the model for WikiSQL using the lookup subset, and for WikiTableQuestions we used the full training set and applied to the lookup subset. For TaPas we used the trained BASE (reset) models[4] for WikiSQL and applied to the lookup subsets of the dev and test sets.

The MRC and MRC$_{\text{xxl}}$ models are based on Machine Reading Comprehension, using the *base v2* and *xxlarge v2* versions of ALBERT. Because this model returns a span rather than a cell prediction, we match each of the top-k span predictions to the closest cell, the cell with the lowest difference in its character offsets. In case multiple of the top-k predictions map to the same cell, these predictions are merged.

We also evaluate the two approaches to RCI: interaction (RCI$_{\text{inter}}$) and representation (RCI$_{\text{repr}}$). Both models use the *base v2* version of ALBERT. Using the *xxlarge v2* ALBERT, we also train another RCI interaction model, RCI$_{\text{xxl}}$. For the representation model we found comparable performance on the column classifier but much lower performance on the row classifier. Therefore the RCI$_{\text{repr}}$ model uses a representation based classifier for columns, while still using the interaction classifier for rows. The RCI$_{\text{inter}}$ model uses interaction classifiers for both rows and columns. Because WikiSQL is the largest dataset by far, for TabMCQ and WikiTableQuestions we first train models on WikiSQL, then fine tune on the target dataset. This gives small but significant gains for TabMCQ but is critical to good performance on WikiTableQuestions.

All models except TaPas produce a ranked list of top-k predictions. We evaluate these predictions using the metrics of Mean Reciprocal Rank (MRR) and Hit@1. Mean Reciprocal Rank is computed by finding the rank of the first correct cell prediction for each question and averaging its reciprocal. If a correct cell is not present in the top-k predictions, it is considered to have an infinite rank. Hit@1 simply measures the fraction of questions that are correctly answered by the first cell prediction.

5.1 Results

Table 3 shows the results on the lookup versions of WikiSQL, TabMCQ, and WikiTableQuestions. Both the interaction and the representation models of RCI outperform all other methods on WikiSQL, TabMCQ, and WikiTableQuestions. Using the representation model for the column classifier reduces performance by less than two percent on WikiSQL, and less than three percent on TabMCQ, but up to seven percent on WikiTableQuestions.

On two of the three datasets both RCI$_{\text{inter}}$ and the more efficient RCI$_{\text{repr}}$ outperform MRC$_{\text{xxl}}$ with far fewer parameters and computational cost. Similarly, RCI with ALBERT-base outperforms even the large version of TaPas trained on WikiSQL, getting 94.6% Hit@1 compared to the 89.43% Hit@1 of TaPas$_{large}$.

	Dev		Test	
Model	**MRR**	**Hit@1**	**MRR**	**Hit@1**
WikiSQL-Lookup				
IS-SP	0.752	67.11%	0.769	69.45%
MRC	0.766	66.91%	0.764	66.52%
TaBERT	0.759	70.78%	0.761	71.16%
TaPas	NA	91.32%	NA	89.02%
RCI$_{\text{inter}}$	**0.963**	**94.48%**	**0.962**	**94.60%**
RCI$_{\text{repr}}$	0.950	92.55%	0.948	92.72%
MRC$_{\text{xxl}}$	0.893	84.89%	0.896	85.33%
TaPas$_{large}$	NA	92.02%	NA	89.43%
RCI$_{\text{xxl}}$	**0.986**	**97.89%**	**0.987**	**97.99%**
TabMCQ-Lookup				
IS-SP	0.375	19.62%	0.301	16.86%
MRC	0.690	60.03%	0.679	59.29%
RCI$_{\text{inter}}$	**0.746**	**67.01%**	**0.742**	**66.26%**
RCI$_{\text{repr}}$	0.727	64.16%	0.725	63.74%
MRC$_{\text{xxl}}$	0.708	63.00%	0.705	62.64%
RCI$_{\text{xxl}}$	**0.758**	**69.10%**	**0.752**	**68.35%**
WikiTableQuestions-Lookup				
IS-SP	0.663	58.87%	0.644	52.69%
MRC	0.681	58.87%	0.601	46.47%
TaBERT	0.686	61.29%	0.646	56.02%
RCI$_{\text{inter}}$	**0.734**	**66.94%**	**0.708**	**61.83%**
RCI$_{\text{repr}}$	0.708	62.90%	0.656	54.77%
MRC$_{\text{xxl}}$	0.783	69.35%	0.732	64.73%
RCI$_{\text{xxl}}$	**0.796**	**72.58%**	**0.794**	**72.20%**

Table 3: Results on TableQA Lookup Datasets

We also compare the performance of the RCI model adapted to aggregation questions to the state-of-the-art TaPas reported results on WikiSQL. We

[4] https://github.com/google-research/tapas

Model	Dev	Test
Wang et al. (2019)	79.4%	79.3%
Min et al. (2019)	84.4%	83.9%
TAPAS$_{large}$	88.0%	86.4%
TABERT	70.53%	70.94%
RCI$_{\text{xxl}}$	**89.7%**	**89.8%**

Table 4: WikiSQL (including aggregation) accuracy

	WikiSQL		TabMCQ		WTQ	
Model	Row	Col	Row	Col	Row	Col
IS-SP	83.1	82.1	70.1	41.5	62.2	82.2
MRC	85.2	73.8	64.6	90.5	56.4	78.8
RCI$_{\text{inter}}$	**96.7**	**98.0**	**73.6**	**92.2**	**64.3**	**92.1**
RCI$_{\text{repr}}$	**96.7**	96.0	**73.6**	89.0	**64.3**	85.1

Table 6: Row/Column Accuracy Results on Test Sets

use the evaluation script provided by TAPAS to produce exactly comparable accuracy numbers for the full WikiSQL dataset. Table 4 shows the RCI model gains over three percent, even without table specific pre-training. It also outperforms TABERT model by a large margin of 18.86%.

In Section 4 we described the method to transform a table into sequence representations of the rows and columns. We do an ablation study on the two larger datasets to understand the impact of incorporating table structure into the sequence representation relative to simply space separating the cell contents. Table 5 shows that we make moderate but significant and consistent gains with this approach, over two percent in Hit@1.

	WikiSQL		TabMCQ	
Model	MRR	Hit@1	MRR	Hit@1
RCI$_{\text{inter}}$	**0.963**	**94.48%**	**0.746**	**67.01%**
-formatting	0.947	92.26%	0.733	64.82%

Table 5: Results on Dev Sets, with formatting ablated

We also decompose the performance of the tested systems in terms of row and column accuracy. The top predicted cell, if wrong, could have the wrong row, the wrong column, or both. Table 6 shows that predicting the correct column is generally easier than predicting the correct row. An interesting exception occurs with MRC on the WikiSQL benchmark: the row prediction is more accurate than the column prediction. For the MRC system, the table is a sequence of column headers, followed by a sequence of rows. Since the table is serialized in row-major order, all of the relevant information for a row is present locally, while the information for columns is distributed through the table sequence representation.

The RCI$_{\text{inter}}$ model is the best at both tasks, with RCI$_{\text{repr}}$ having the same performance at the row level task, since it uses the same model for rows. The TabMCQ column level performance of MRC is within two percent of RCI$_{\text{inter}}$, which may be surprising, especially considering its performance on WikiSQL. TabMCQ tables are constructed in an unusual way that permits high column prediction performance for an MRC system. The rows in TabMCQ have the structure of sentences, which is helpful for a system trained on the SQuAD and NQ reading comprehension tasks (Refer to the appendix for a sample TabMCQ table).

5.2 Error Analysis

To better understand the advantages and disadvantages of the Row-Column Intersection approach, we examine the 20 cases in the dev set of WikiTableQuestions-Lookup where RCI$_{\text{inter}}$ does not provide the correct answer in first position but MRC$_{\text{xxl}}$ does. We find nine cases where we could identify nothing that in principle prevents the RCI$_{\text{inter}}$ model from answering correctly. We find seven cases where multiple rows need to be considered together, while the RCI models always consider rows independently. WikiTableQuestions includes some questions like Table 7. Although the answer to this question is a cell in the table, it requires something like aggregation to answer. All rows for a given year must be checked to see if there is a ‘1st’ in the Place column. This violates a key assumption of RCI: that rows may be examined independently. The final four cases also violate the assumptions of RCI. In two cases the answer is in the header of the table, while RCI assumes that it will be a cell. In one case the table extraction failed, and in the final case the question asks about the string length of one of the columns where the answer (8) happens to be in the table.

We also examine the cases where MRC$_{\text{xxl}}$ does not find the correct answer in first position but RCI$_{\text{inter}}$ does. The most frequent error, occurring in eight of the seventeen cases, is a ‘near-miss’. Either MRC$_{\text{xxl}}$ chooses a value from the wrong column in the right row or a value from the row before or after. This is illustrated in Table 8, where MRC$_{\text{xxl}}$ selects a value near the desired date that

Season	...	Discipline	Place
2012		Downhill	2nd
2012		Downhill	3rd
...			
2013		Downhill	3rd
2013		Super-G	1st
...			
2014		Super-G	2nd
2014		Super-G	1st
...			

In which year did Tina Weirather not earn 1st place?
Answer: 2012

Table 7: Example Multiple-Row Question

is easily confused with a location. In other cases a location from the previous or next row, which are adjacent in the input passage, can be selected instead.

Date	Opponent	Venue
	...	
27 Aug 2005	Wigan Athletic	JJB Stadium
10 Sept 2005	Chelsea	Stamford Bridge
17 Sept 2005	West Bromwich Albion	Stadium of Light
25 Sept 2005	Middlesbrough	Riverside Stadium
1 Oct 2005	West Ham United	Stadium of Light
	...	

Where was the match on 17 September 2005 played?
Answer: Stadium of Light
MRC$_{\text{xxl}}$ Answer: West Bromwich Albion

Table 8: Example of Near-Miss by MRC

We also conduct an error analysis of RCI$_{\text{xxl}}$ on the first 50 aggregation questions it misses on the dev set of WikiSQL. The largest category, with 24 cases, is correct answers by RCI$_{\text{xxl}}$ counted wrong by mistakes in the ground truth. Usually (23) the ground truth indicates that there should be COUNT aggregation when no aggregation is correct. For example, “What is the rank of manager Rob Mcdonald?” where *Rank* is one of the table columns is mistakenly indicated as a COUNT aggregation question.

The second largest category (9) occurs when the cells are ranked correctly, and the correct aggregation is predicted, but the threshold for choosing the cells to aggregate is too low (1) or too high (8).

Another common error (7) occurs when RCI$_{\text{xxl}}$ predicts a lookup question with the answer in a similar numeric column when aggregation is required. For example, the question “How many votes were taken when the outcome was "6th voted out day 12"?” is asked for a table with a *Votes* column. RCI$_{\text{xxl}}$ predicts it as a lookup question with the answer (“2-2-1 3-0”) from this column, while the ground truth is a COUNT aggregation.

The final significant category (7) is cases of questions that are unanswerable. This can occur because the table does not contain an answer or because the answer cannot be computed from a SQL query, such as when the answer is a sub-string of a cell.

The final three error cases are: a wrong column is selected (the episode number in series rather than the episode number in season); the question “What is the result when the 3rd throw is not 8?” is interpreted as “What is the result when the 3rd throw is *something other than* 8?” rather than the ground truth “What is the result when the 3rd throw is *literally ‘not 8’*?”; and non-Latin characters must be matched to select the correct row.

6 Conclusion

In this paper we propose two novel techniques, RCI interaction and RCI representation, to tackle the problem of locating answers over tables for given natural language questions. These transformer based models are fine-tuned on ground truth tables to predict the probability of containing the answer to a question in the rows and columns of tables independently. These probabilities are either used to answer questions directly or highlight the relevant regions of tables as a heatmap, helping users to easily locate the answers over tables.

Our experiments prove that the RCI model outperforms the state-of-the-art transformer based approaches pre-trained on very large table corpora (TaPas (Herzig et al., 2020) and TaBert (Yin et al., 2020)), achieving $\sim$3.4% and $\sim$18.86% additional precision improvement on the standard WikiSQL benchmark including both Lookup and Aggregation questions. The representation model, on the other hand, enables pre-processing the tables and producing the embeddings to store and further use during online query processing, providing significant efficiency advantages without compromising much on the accuracy of finding cell values in tables. As for the future work, we plan to explore the exploitation of domain-specific taxonomies and embeddings generated for domain-specific corpora to tackle the problem of answering natural language questions over tables in domains such as finance, aviation and health care.

References

Chris Alberti, Daniel Andor, Emily Pitler, Jacob Devlin, and Michael Collins. 2019a. Synthetic QA corpora generation with roundtrip consistency. *CoRR*, abs/1906.05416.

Chris Alberti, Kenton Lee, and Michael Collins. 2019b. A BERT baseline for the natural questions. *arXiv preprint arXiv:1901.08634*, pages 1–4.

Eric Brill, Susan Dumais, and Michele Banko. 2002. An analysis of the AskMSR question-answering system. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, pages 257–264. Association for Computational Linguistics.

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):538–549.

Mustafa Canim, Cristina Cornelio, Arun Iyengar, Ryan Musa, and Mariano Rodriguez. 2019. Schemaless queries over document tables with dependencies. *CoRR*, abs/1911.09356.

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*, pages 817–828.

Pradeep Dasigi, Matt Gardner, Shikhar Murty, Luke Zettlemoyer, and Eduard Hovy. 2019. Iterative search for weakly supervised semantic parsing. 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)*, pages 2669–2680, Minneapolis, Minnesota. Association for Computational Linguistics.

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)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Matt Gardner, Dirk Groeneveld, Brendan Roof, Pradeep Dasigi, Michael Schmitz, and Nitish Gupta. 2020. allennlp-semparse. `https://github.com/allenai/allennlp-semparse`.

Michael R. Glass, Alfio Gliozzo, Rishav Chakravarti, Anthony Ferritto, Lin Pan, G. P. Shrivatsa Bhargav, Dinesh Garg, and Avirup Sil. 2020. Span selection pre-training for question answering. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 2773–2782. Association for Computational Linguistics.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. 2020. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8342–8360, Online. Association for Computational Linguistics.

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

Sujay Kumar Jauhar, Peter Turney, and Eduard Hovy. 2016. Tabmcq: A dataset of general knowledge tables and multiple-choice questions.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466.

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 *International Conference on Learning Representations*.

Jimmy Lin. 2007. An exploration of the principles underlying redundancy-based factoid question answering. *ACM Transactions on Information Systems (TOIS)*, 25(2):6–es.

Sewon Min, Danqi Chen, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2019. A discrete hard EM approach for weakly supervised question answering. 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)*, pages 2851–2864, Hong Kong, China. Association for Computational Linguistics.

Lin Pan, Rishav Chakravarti, Anthony Ferritto, Michael Glass, Alfio Gliozzo, Salim Roukos, Radu Florian, and Avirup Sil. 2019. Frustratingly easy natural question answering.

Marius Paşca. 2003. Open-domain question answering from large text collections. *Computational Linguistics*, 29(4).

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the*

7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 1470–1480, Beijing, China. Association for Computational Linguistics.

Rakesh Pimplikar and Sunita Sarawagi. 2012. Answering table queries on the web using column keywords. *Proceedings of the VLDB Endowment*, 5(10):908–919.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. *arXiv preprint arXiv:1806.03822*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Roee Shraga, Haggai Roitman, Guy Feigenblat, and Mustafa Canim. 2020. Ad hoc table retrieval using intrinsic and extrinsic similarities. In *WWW '20: The Web Conference 2020*, pages 2479–2485. ACM / IW3C2.

Huan Sun, Hao Ma, Xiaodong He, Wen-tau Yih, Yu Su, and Xifeng Yan. 2016. Table cell search for question answering. In *Proceedings of the 25th International Conference on World Wide Web*, pages 771–782.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008. Curran Associates, Inc.

Petros Venetis, Alon Halevy, Jayant Madhavan, Marius Pasca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. 2011. Recovering semantics of tables on the web. *Proceedings of the VLDB Endowment*, 4(9).

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*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Bailin Wang, Ivan Titov, and Mirella Lapata. 2019. Learning semantic parsers from denotations with latent structured alignments and abstract programs. 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)*, pages 3774–3785, Hong Kong, China. Association for Computational Linguistics.

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*, pages 97–108.

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*, pages 8413–8426, Online. Association for Computational Linguistics.

Meihui Zhang and Kaushik Chakrabarti. 2013. Infogather+ semantic matching and annotation of numeric and time-varying attributes in web tables. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 145–156.

Shuo Zhang and Krisztian Balog. 2018. Ad hoc table retrieval using semantic similarity. In *Proceedings of WWW 2018*, WWW '18, page 1553–1562.

Zhuosheng Zhang, Yuwei Wu, Junru Zhou, Sufeng Duan, and Hai Zhao. 2020a. Sg-net: Syntax-guided machine reading comprehension. *AAAI*.

Zhuosheng Zhang, Junjie Yang, and Hai Zhao. 2020b. Retrospective reader for machine reading comprehension. *arXiv preprint arXiv:2001.09694*.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

A Appendix

Both the MRC and RCI training was carried out using the pytorch transformers toolkit made available by Huggingface[5]. Table 9 gives the number of parameters for each introduced model.

Model	Parameters
MRC	25M
MRC$_{\text{xxl}}$	235M
RCI$_{\text{inter}}$	50M
RCI$_{\text{repr}}$	75M

Table 9: Number of parameters for introduced models

A.1 MRC **Model Training**

Models were trained and decoded using single GPU training on machines with 32GB Tesla V100 GPUs with 16-bit precision. This results in processing

[5] `https://github.com/huggingface/transformers`

Hyperparameter	Setting
Max query tokens	64
Max answer tokens	30
Batch size	32
Optimizer ϵ β_1 β_2	Adam 1^{-8} 0.9 0.999
Warmup ratio %	10%
Learning rate warmup	Linear
Peak learning rate	3e-5
Epochs	3
Document stride	128
Max sequence length	512

Table 10: SQ2 MRC Hyperparameter Configurations

speeds of $\sim 35-75$ features per second (depending on whether it's MRC or MRC$_{\text{xxl}}$). Note: due to the 512 sequence length limitation, each query-table example may be split into multiple feature vectors as per (Devlin et al., 2019) and subsequent work in building MRC models using transformer networks.

SQuAD 2.0 (SQ2) Training

Table 10 lists the hyperparameters used in line with (Lan et al., 2020) for training MRC on SQ2.

Natural Questions (NQ) Training

Table 11 lists the hyperparameters used in line with (Pan et al., 2019; Alberti et al., 2019b) for training MRC on NQ. Note that the NQ training requires a few modifications from the default SQuAD style QA task to account for the dataset's additional answer types $[short, long, yes, no, null]$[6]:

1. A dense layer ($\mathbf{W}_3 \in \mathbb{R}^{5\times D_e}$) is added which operates only on the contextualized representation of the $[CLS]$ token to produce a likelihood prediction for each answer type: $\boldsymbol{\alpha}_l = softmax(\mathbf{W}_3\mathbf{h}_{[CLS]})$).

2. At training time an additional cross entropy loss term is added between the true answer type labels and the predicted answer type likelihoods ($\boldsymbol{\alpha}_l$).

3. At prediction time the final score is a simple weighted average of the short answer likelihood score ($\boldsymbol{\alpha}_{l=short}$) and the SQUAD like max span score based on $\boldsymbol{\alpha}_b^{t_j}$ and $\boldsymbol{\alpha}_e^{t_i}$.

Hyperparameter	Setting
Max query tokens	18
Max HTML spans (top level)	48
Max answer tokens	30
Batch size	48
Optimizer ϵ β_1 β_2	Adam 1^{-8} 0.9 0.999
Warmup ratio	10%
Learning rate warmup	Linear
Peak learning rate	1.6E-5
Weight decay	0.01
Gradient clipping (norm)	1.0
Epochs	1 (All Questions) + 1 (Table Questions)
Document stride	192
Max sequence length	512
Negative Subsampling (Answerable Questions)	4%
Negative Subsampling (Un-answerable Questions)	1%

Table 11: NQ MRC Hyperparameter Configurations

Table Data Sets

An additional epoch was trained using the same configurations as in table 11 using the datasets discussed in the evaluation section.

A.2 RCI Model Training Hyperparameters

All models were trained with the same settings for Adam, Gradient clipping and Weight decay. The max sequence length for interaction models was 512, while for representation models it was 256 for both question and column sequence representation. We did not test variations on these hyperparameters. Table 12 gives the constant hyperparameters and the range tested for the others.

We manually tuned learning rate (LR), batch size, number of training epochs (E), and fraction of training instances for warmup (Warm) on the development set for each dataset. We also report the number of development runs for each model. The final hyperparameters were selected as those that maximize ROC on the row or column subtask. Table 13 shows these hyperparameters. The 93%

[6] As in (Alberti et al., 2019b), *yes* and *no* questions are not handled by the model as they are less than 2% of the data and *long* answers are predicted by looking up the top level HTML span which contains the predicted *short* span.

Hyperparameter	Setting
Optimizer ϵ β_1 β_2	Adam 1^{-8} 0.9 0.999
Learning rate warmup	Linear
Weight decay	0.01
Gradient clipping (norm)	1.0
Max sequence length interaction representation	 512 256

Hyperparameter	Range
Learning Rate	2.5e-6 to 5e-5
Batch Size	64, 128
Epochs	2 to 6
Warmup	0% to 93%

Table 12: RCI Hyperparameter Ranges

Model	LR	Batch	E	Warm	Runs
WikiSQL					
Row	5e-5	128	2	0.93	8
Col$_{inter}$	2e-5	64	3	0.13	4
Col$_{repr}$	1e-5	64	4	0.11	9
TabMCQ					
Row	1e-5	64	2	0.02	7
Col$_{inter}$	4e-5	64	4	0.09	8
Col$_{repr}$	1e-5	64	6	0.10	3
WikiTableQuestions					
Row	5e-5	128	2	0.00	3
Col$_{inter}$	1e-5	64	4	0.00	2
Col$_{repr}$	1e-5	64	6	0.00	4

Table 13: RCI Tuned Hyperparameters

warmup instances for WikiSQL Row was selected by adding one too many zeros to the number of warmup instances while aiming for around 10% warmup. However, this turned out to be the best run.

Training was done on a single machine with four P100 GPUs. For all models, training time was between 70 and 80 instances per second, giving training times for WikiSQL of around two and a half hours per epoch for rows and one hour per epoch for columns.