An Inner Table Retriever for Robust Table Question Answering

Weizhe Lin[*2], **Rexhina Blloshmi**[1],
Bill Byrne[1,2], **Adrià de Gispert**[1], **and Gonzalo Iglesias**[1]
[1]Amazon Alexa AI
[2]University of Cambridge
wl356@cam.ac.uk {blloshmi, willbyrn, agispert, gjii}@amazon.com

Abstract

Recent years have witnessed the thriving of pre-trained Transformer-based language models for understanding semi-structured tables, with several applications, such as Table Question Answering (TableQA). These models are typically trained on joint tables and surrounding natural language text, by linearizing table content into sequences comprising special tokens and cell information. This yields very long sequences which increase system inefficiency, and moreover, simply truncating long sequences results in information loss for downstream tasks. We propose Inner Table Retriever (ITR),[1] a general-purpose approach for handling long tables in TableQA that extracts sub-tables to preserve the most relevant information for a question. We show that ITR can be easily integrated into existing systems to improve their accuracy with up to 1.3-4.8% and achieve state-of-the-art results in two benchmarks, i.e., 63.4% in WikiTableQuestions and 92.1% in WikiSQL. Additionally, we show that ITR makes TableQA systems more robust to reduced model capacity and to different ordering of columns and rows.

1 Introduction

Tables offer a systematic way of storing information in the Web. Extracting information from Web tables poses different challenges than extracting information from relational databases with logical queries, especially when queried via Natural Language (NL) user questions. Table Question Answering (TableQA) is the task of answering such questions with *factoid* answers extracted from table content. This requires developing models with the ability to reason over and understand tables. Following the success of the pre-training paradigm for understanding NL text (Devlin et al., 2019), some recent research has focused on pre-training Transformer models (Vaswani et al., 2017)

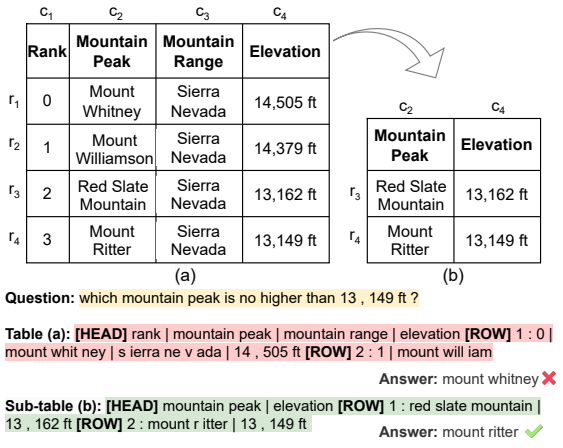

Figure 1: TableQA example with the model input length budget set to 50 tokens using TaPEx tokenization and table linearization format; (a) is an *overflow* table because the linearized version must be truncated. Our method can identify sub-tables like (b) within the length budget, removing the information loss.

on large corpora of *linearized* tables in a self-supervised fashion using encoder-only architectures (Herzig et al., 2020; Yin et al., 2020; Yang et al., 2022), or encoder-decoder architectures (Liu et al., 2022; Jiang et al., 2022). These so-called Tabular Language Models (Dong et al., 2022, TaLMs) were fine-tuned on the TableQA downstream task—among others—to achieve state-of-the-art performance (Herzig et al., 2020; Liu et al., 2022). However, the self-attention mechanism in TaLMs has a quadratic complexity on the dimensionality of input tables, which might consist of tens or even hundreds of rows and columns, thus yielding longer sequences than TaLMs can easily handle. State-of-the-art TableQA models handle this limitation by truncating the linearized table to fit an input length budget, e.g., of 512 and 1024 tokens for Herzig et al. (2020, TaPas) and Liu et al. (2022, TaPEx), respectively. In other applications, simple sequence truncation might be reasonable, e.g., encoding only the initial paragraphs of a Wikipedia document pre-

[*]Work done as an intern at Amazon Alexa AI.
[1]We make our code available at: https://github.com/amazon-science/robust-tableqa

suming it comprises a summary, or dropping earlier turns in Conversational QA to focus on the recent ones. However, in TableQA it is not realistic to assume that relevance depends on the position within the linearized sequence, especially because different questions require various table regions to be properly answered. For example, even for a standard dataset such as WikiTableQuestions, naive truncation allows information loss affecting 18.1%-44.9% of tables, which limits QA accuracy (see §4.1 for more details). This is also an important limitation in latency-constrained realistic use cases that use big tables, while limiting Transformer models even down to 64 input tokens. To this end, a content-driven strategy is needed to avoid information loss. We refer to tables exceeding the input length budget as *overflow* tables, as opposed to *compact* tables, which fit within the budget. An example of an overflow table is shown in Figure 1(a), where naive truncation leads to the wrong answer.

In this work, we propose ITR to improve on this problem by creating smaller sub-tables, i.e., within a length budget, based on dense retrieval of table rows and columns according to the relevance to the question. An example is shown in Figure 1(b). Our method is flexible and can be integrated *off-the-shelf* into virtually any existing TableQA system. To the best of our knowledge, our work is the first to propose a neural-based sub-table selection in the context of TableQA that improves denotation accuracy (Pasupat and Liang, 2015), especially for the overflow tables, setting a new state of the art. Other input selection strategies, mainly heuristics-based, have also been proposed in the literature (Krichene et al., 2021; Yin et al., 2020; Eisenschlos et al., 2021), which we discuss further in Section 2.

To summarize, the contributions of this work are the following:

1. We propose ITR, an efficient approach to handling overflow tables for TableQA models, which produces sub-tables containing the most relevant information for answering a question while fitting within the budget.

2. We combine ITR with existing TableQA systems such as TaPas, TaPEx, and OmniTab (Jiang et al., 2022), and achieve a new state of the art for two standard benchmarks, WikiSQL and WikiTableQuestions.

3. We evaluate the robustness of ITR against current TableQA models on *overflow* tables, when reducing the length budget, and when repositioning relevant table information.

2 Related Work

The works most related to ours employed different pruning strategies to handle large tables. Yin et al. (2020, TaBERT) introduce the concept of *content snapshot* as encoder input. This snapshot is composed of a small number of rows which are chosen based on the n-gram similarity between the question and column headers and cell values. In a similar fashion, Eisenschlos et al. (2020) explore Jaccard similarity to obtain the most similar columns. In addition, they leverage model tokenizer to reduce cell tokens to their first token, when necessary, and dropping entire rows that do not fit the length budget. However, lexical similarity and naive truncation are not flexible and lead to information loss, which has a drastic effect on TableQA performance, as we show in our experiments. Another line of work focuses on balancing model efficiency and accuracy when handling long tables. Krichene et al. (2021, DOT) first uses a smaller *pruning* transformer to selects top-K tokens from the input table, and then a larger second task-specific transformer takes into consideration only the selected K tokens and their pruning scores; Eisenschlos et al. (2021, MATE) can accept more tokens while not significantly increasing latency. Authors apply sparse self-attention and use different attention heads for tokens in the same column and row. However, their proposed mechanisms are deeply integrated with the task-specific model and must be jointly trained. In contrast, our ITR method, as a flexible plug-in process, can work independently of any underlying TableQA model. Moreover, our approach is complementary to theirs since ITR can drop irrelevant information in tables efficiently and pass the trimmed compact table to the underlying model. This can further exploit the potentials of virtually any TableQA models and improves their performance. Although not specific to TableQA, works such as Wang et al. (2021) and Chen et al. (2021) employ chunking strategies, i.e., encoding table chunks separately and then aggregate them together. However chunking is not widely employed in the literature (Dong et al., 2022), due to encoding overhead requiring multiple inference calls for each chunk.

Algorithm 1 Creating N sub-tables from T for q.

1: **def** ITR(q, T, E_Q, E_T, N, b):
2: $Z \leftarrow [], T_{sub} \leftarrow [], T_{aux} \leftarrow [], L \leftarrow []$
3: $e_q \leftarrow E_Q(q)$
4: **for each** $i \in \texttt{Items}(T)$ **do**
5: $e_t \leftarrow E_T(i)$
6: $Z \leftarrow Z \cup \{\text{sim}(e_q, e_t), i\}$
7: **for each** $i \in \texttt{Sorted}(Z)$ **do** ▷ ↓ sim
8: $T_{aux} \leftarrow T_{aux} \cup i$
9: **if** $\texttt{CheckValid}(T_{aux})$ **then**
10: $T_{sub} \leftarrow T_{sub} \cup T_{aux}$
11: **for each** $t \in T_{sub}$ **do**
12: $L \leftarrow L \cup \{\texttt{Length}(t), t\}$
13: $T_{sub} \leftarrow []$
14: **for each** $(l, t) \in \texttt{Sorted}(L)$ **do** ▷ ↓ length
15: **if** $l > b$ **then** continue
16: $T_{sub} \leftarrow T_{sub} \cup t$
17: **return** $T_{sub}[:N]$

3 Methodology

3.1 Task

Given a question q and a table T, TableQA systems return an answer denotation a, either by performing table cell selection or as the result of operations (such as counting) carried out over an aggregation of table cells. This work aims to find one or more sub-tables T_{sub} containing the most relevant information from T needed to answer q; T_{sub} can replace T as the input to virtually any existing TableQA system.

To this end, we map a table T into a set of items, where an item is either a complete row or a complete column. For an $n \times m$ table, this gives a set of items $\{r_1, \ldots, r_n, c_1, \ldots, c_m\}$. Then, we construct sub-tables by specifying subsets of these rows and columns: a sub-table consists of the cells at the intersection of the selected rows and columns. We refer to each such set of rows and columns as a *mix*, and note that a mix must contain at least one row and one column to specify a valid sub-table. The table in Figure 1(a) is defined as $\{r_1, r_2, r_3, r_4, c_1, c_2, c_3, c_4\}$, and the sub-table in Figure 1(b) is specified as the mix $\{r_3, r_4, c_2, c_4\}$. $\{c_2, c_4\}$ is not a valid sub-table, since no cells will be intersected with only column-wise items.

3.2 Inner Table Retriever

ITR is a process of retrieving table rows and columns, and combining them to form sub-tables T_{sub}, with q as a query. We describe the steps for creating sub-tables in Algorithm 1. Lines 2-6 compute item similarities to q, and the function $\texttt{Items}(T)$ in Line 4 maps T into its $n + m$ items. Following Karpukhin et al. (2020, DPR), we have two fine-tuned encoders, one for questions (E_Q) and another for table items (E_T).

We fine-tune DPR encoders using a question as a query and the row/columns that contain the gold answer cells as positive items. Then we sample the negative items from the remaining row/columns in the table. We leverage the standard DPR contrastive loss to fine-tune the two encoders so that the similarities between question and positive item embeddings are maximized.[2] At inference time, we compute the contextual embeddings for the question (Line 3) and all the items in a table (Line 5) and compute their similarity, sim($\cdot$), as the dot product in Line 6. In practice, pre-computed embeddings of table items can be cached offline.

Creating sub-tables. In Lines 7-10 we loop through the table items ranked by highest similarity and aggregate in T_{aux}. $\texttt{CheckValid}(T_{aux})$ verifies that T_{aux} is a valid sub-table, i.e., there exists at least one column-row intersection (see §3.1).

Choosing the most appropriate sub-tables. In Lines 11-16 we sort the sub-tables by their sequence length in descending order, and filter out any sub-table which exceeds the length budget b. Finally, in Line 17 we return top-N remaining largest sub-tables that fit the budget length to be used as input to the TableQA model. Each returned sub-table is guaranteed to contain the most relevant items, due to the sorting operation in Line 7.

3.3 TableQA with ITR

Through the ITR process, we obtain N sub-tables T_{sub} as replacement for the original table T. Each sub-table, together with the associated NL question q, is linearized into a sequence of tokens prior to encoding, using the corresponding TableQA tokenizer. We exemplify this in Figure 1, where both table (a) and sub-table (b) are similarly processed. Each linearized sequence is used as input to the TableQA model, thus obtaining N predictions, out of which we choose the most confident answer. We empirically find that the model performance is marginally better when $N > 1$, instead of only considering the largest sub-table (see §6; Appendix C).

[2] More details in Appendix A.1.

	WikiSQL		WikiTQ	
max tokens	Dev	Test	Dev	Test
1024	6.8	9.7	19.2	18.1
512	30.6	32.7	44.4	44.9
256	68.6	69.9	81.5	82.6
128	98.0	98.2	100	100
64	100	100	100	100
total samples	8,421	15,878	2,831	4,344

Table 1: Total number of samples and overflow rate (%) for different length budgets in WikiSQL and WikiTQ. Question-table pair sequence length is calculated based on TaPEx's tokenizer and linearization strategy.

4 Experimental Setup

4.1 Datasets and Evaluation

We use two popular datasets for TableQA which are constructed from Wikipedia: WikiSQL[3] (Zhong et al., 2017) and WikiTableQuestions[4] (Pasupat and Liang, 2015, WikiTQ), with each posing different challenges. WikiSQL is a simpler TableQA dataset than WikiTQ as it requires mainly filtering and simple operations of table cells to answer a question. WikiTQ demands more complicated reasoning capabilities such as aggregations, comparisons, superlatives, or arithmetic operations, e.g., SUM, MAX. We measure Denotation Accuracy (DA) for validation and test sets, to assess whether the predicted answer(s) is equal to the ground-truth answer(s). Additionally, we introduce the distinction between *compact* tables and *overflow* tables, which is determined by the length of linearized question-table pair. Statistics in Table 1 show that even when using a relatively high number of tokens, i.e., 1024 or 512—the allowed maximum supported by most of Transformer-based encoders— the range of overflow tables is very high, 7-33% in WikiSQL, and 18-45% in WikiTQ. ITR is applied only for overflow tables, as the compact ones already fit in the token budget. Finally, we evaluate ITR retrieval ability on WikiSQL using Recall@K, which measures whether *all* the gold rows/columns for answering a question are among the top-K retrieved items.

4.2 Training Setup

For both, the ITR retrieval component and the underlying TableQA systems that we train in-house, we choose the best checkpoint based on the performance on the validation set. Otherwise, for TableQA systems in the literature, we use the released checkpoints from `huggingface`.

Since WikiTQ does not provide SQL annotations, which are needed to obtain gold answer cell coordinates for supervising the ITR retriever (see §3.2), we use the model trained on WikiSQL in a zero-shot fashion to retrieve the relevant table items for the WikiTQ TableQA task. We set the number of sub-tables N=10 when using TaPEx and OmniTab systems, and N=1 with TaPas (see Appendix C). We provide details and hyperparameters for ITR and TableQA models in Appendix A.

4.3 Comparison Systems

We evaluate the effectiveness of ITR by comparing our ITR-enhanced baselines with recent models from literature that report performance on WikiSQL or WikiTQ (Min et al., 2019; Herzig et al., 2020; Yu et al., 2021; Liu et al., 2022; Jiang et al., 2022). We provide detailed comparisons using TaPEx, TaPas, and OmniTab, with ITR included in inference alone, as well with ITR integrated into TaPEx training. TaPEx leverages BART (Lewis et al., 2020), an encoder-decoder model, and fine-tunes it to act as a SQL executor. For the TableQA downstream task, TaPEx takes as input a <question, table> pair and autoregressively generates an answer, implicitly performing any kind of aggregations. OmniTab extends TaPEx and leverages multi-tasking and data augmentation to establish a new state of the art in WikiTQ. TaPas is an encoder model built on top of BERT (Devlin et al., 2019) which takes as input a <question, table> pair and learns jointly to: i) select table cells by thresholding the cell confidence, and ii) predict an explicit operator to aggregate results from the cell selection. We use a recent version of TaPas proposed by Eisenschlos et al. (2020) which leverages intermediate pretraining and table pruning, e.g., cell truncation to the first token and dropping rows that exceed the limit, improving significantly the initially released model (TaPas$_{v0}$). When applying ITR, we disable TaPas table pruning and use the full sub-table(s) (see Appendix D for a case study). Regarding their capacity, TaPEx and OmniTab support up to 1024 tokens, while TaPas can take up to 512 tokens.

It is worth noting that we have not been able to fully reproduce the reported results in the literature of TaPas, TaPEx and OmniTab with their

[3] https://huggingface.co/datasets/wikisql

[4] https://huggingface.co/datasets/wikitablequestions

Models	Dev	Test
Min et al. (2019)	84.4	83.9
TaPas$_{v0}$ (Herzig et al., 2020)	85.1	83.6
TaPas (Eisenschlos et al., 2020)	89.8	-
Yu et al. (2021)	85.9	84.7
TaPEx (Liu et al., 2022)	89.2	89.5
OmniTab (Jiang et al., 2022)	-	88.7
ITR → TaPEx (#6)	91.8	91.6
ITR → TaPas (#5)	**92.1**	**92.1**

Table 2: Results on WikiSQL. **Bold** denotes the best DA for each split. # references a row in Table 4.

Models	Dev	Test
TaPas$_{v0}$ (Herzig et al., 2020)	29.0	48.8
TaPas (Eisenschlos et al., 2020)	50.9	-
Yin et al. (2020)	53.0	52.3
Yu et al. (2021)	51.9	52.7
TaPEx (Liu et al., 2022)	57.0	57.5
OmniTab (Jiang et al., 2022)	-	62.8
ITR → TaPEx (#9)	61.8	61.5
ITR → TaPas (#5)	50.5	50.8
ITR → OmniTab (#7)	**62.1**	**63.4**

Table 3: Results on WikiTQ. **Bold** denotes the best DA for each split. # references a row in Table 4.

`huggingface` implementations, due to cross-framework dependencies or any model-specific preprocessing or evaluation scripts. To assess the realistic contribution of ITR, we also report reproduced results using the same unified framework across all models. We discuss reproducibility in Appendix A.3.

5 Results

5.1 Main Results

We report the performance of our best ITR-enhanced TableQA models for WikiSQL and WikiTQ in Tables 2 and 3, respectively, and compare with the state-of-the-art results as reported in the literature. In WikiSQL, ITR-enhanced models consistently outperform all previous baselines. ITR improves TaPEx and TaPas performance with 2.6 and 2.3 DA points in the Dev set, respectively. ITR → TaPas sets a new state of the art for WikiSQL, reaching a DA of 92.1% in the Test set. In WikiTQ, results are mixed: ITR shows slight degradation over TaPas in the Dev set. Combined with TaPEx, ITR improves DA by 4 points. Further, combined with OmniTab, ITR improves DA by 0.6%, reaching a new state-of-the-art result on this task.

In Table 4 we compare ITR in a unified experimentation setting using the released model checkpoints in `huggingface` for inference-only evaluation, and the in-house fine-tuned TaPEx model that enables both training and evaluation with ITR. For OmniTab, only the WikiTQ model is made available, thus we do not evaluate in WikiSQL. We also break down the performance into compact and overflow tables to better assess the contribution of ITR. When evaluated using the same settings, ITR consistently improves on top of baselines(#5-7 *versus* #1-3) on WikiSQL and WikiTQ, respectively: TaPas (+2.6% and +0.2%), TaPEx (+2.9% and +1.4%) and OmniTab (+1.3% on WikiTQ only). This is also true for the in-house trained TaPEx (#4 *versus* #8). ITR reduces the compact/overflow performance gap significantly: ITR increases overflow DA with up to 30% (with TaPEx) and 9.4% (with OmniTab) on WikiSQL and WikiTQ, respectively, making the underlying model more robust to larger tables. In fact, in WikiSQL we close the gap between compact and overflow tables (#5). These results show that ITR can be applied at inference time to always improve performance, by presenting more complete and relevant information in state-of-the-art model decision process, even without interfering with the model training process. Additionally, fine-tuning TaPEx with ITR sub-tables (#9) is better than plugging ITR only at inference time (#8). Training with ITR improves the *overflow* DA by 1.7% on WikiSQL and 3.7% on WikiTQ. The Test performance increases by 0.8% and 2.7%, respectively. Not only does ITR improve the decision process of underlying models at inference time, but it further increases performance if included in the model training process as well. This demonstrates that ITR can be flexibly applied to any TableQA model.

5.2 Repositioning Denotations

In Table 4 we notice that models that naively truncate table sequences (#1-4) may observe the correct denotations within the length budget, achieving 58.2-63.2% DA in WikiSQL even for overflow samples. This is because the original table happens to present the answer early, e.g., in the first column/row. To fully investigate the potential and usefulness of ITR, we create an extreme case by moving gold rows and columns altogether to the bottom right of the table, thus reducing the chances of arbi-

#	Models	WikiSQL				WikiTQ			
		Dev	Test	Compact	Overflow	Dev	Test	Compact	Overflow
1	TaPas (hf)	90.4	89.5	92.1	76.3	50.2	50.6	55.4	37.6
2	TaPEx (hf)	89.5	88.7	91.9	59.1	57.2	55.5	60.5	32.7
3	OmniTab (hf)	-	-	-	-	61.0	62.1	**66.5**	35.9
4	TaPEx	88.4	87.7	90.8	58.2	58.7	57.8	62.8	35.3
5	ITR → TaPas (hf)	**92.1**	**92.1**	**92.1**	**92.1**	50.5	50.8	55.4	38.2
6	ITR → TaPEx (hf)	91.8	91.6	91.9	89.1	58.4	56.9	60.5	41.1
7	ITR → OmniTab (hf)	-	-	-	-	**62.1**	**63.4**	**66.5**	**45.3**
8	ITR → TaPEx	90.5	90.6	90.8	87.7	60.4	58.8	62.8	41.1
9	ITR → TaPEx *(+train)*	91.3	91.4	91.6	89.4	61.8	61.5	65.2	44.8

Table 4: DA on WikiSQL and WikiTQ when applying ITR at inference time only or also in training. TaPEx denotes the in-house fine-tuned TaPEx (hf) model. Compact and Overflow are subsets of the Test split. Only Dev/Test DA values are directly comparable across models. This is because the token limit is different for TaPEx and OmniTab (1024) and TaPas (512), and overflow samples are of different sizes for these models, i.e., 9.7% for TaPEx and OmniTab (see Table 1) and 16.6% for TaPas. **Bold** denotes the best accuracies for each dataset split.

Models	Test	Test$_{ext.}$	Compact	Compact $_{ext.}$	Overflow	Overflow$_{ext.}$
TaPEx	87.7	82.8	90.8	89.1	58.2	23.6
ITR → TaPEx	91.4	90.0	91.6	90.5	89.4	85.3
ITR → TaPEx *(+train shuffled)*	91.5	**90.8**	91.8	**91.2**	89.0	**87.1**

Table 5: DA in WikiSQL when repositioning denotations to the bottom-right corner of the table, denoted as D$_{ext.}$ where D is any of the dataset splits. **Bold** denotes the best DA for each dataset split in the extreme scenario.

trary success due to dataset design. For this analysis, we use WikiSQL since it provides the gold cell annotations. We evaluate TaPEx in combination with ITR on both the original and extreme-case set and report results in Table 5. Unsurprisingly, the overall performance of TaPEx drops sharply from 87.7% to 82.8%, with the overflow accuracy dropping from 58.2% to only 23.6%. This suggests that in extreme cases failure to address the long table issue will harshly degrade model performance and that the reported DA values can be optimistic due to the convenient position of the relevant information on top of the table. ITR-enhanced TaPEx is less affected: the overall DA degrades by 1.4% and overflow accuracy by only 4.1% as opposed to 34.6% drop of TaPEx. The 4.1% drop is mainly due to the positioning bias that the TableQA system might have learned during training. In fact, if we introduce row/column repositioning also at training time *(+train shuffled)*, i.e., making gold answer denotations appear equally in any possible position of the input sub-table, we observe a reduced impact of the extreme repositioning at inference time: the performance drop is only 0.7% overall and 1.9% in the overflow split. We discuss row/column positioning effects in Appendix B.

5.3 Reducing the Input Length Budget

In production pipelines, latency is a crucial issue and 1024 tokens can be unrealistic in achieving a smooth user experience. We explore the input length budget within 64 to 1024 tokens and compare TaPEx and TaPas when combined with ITR in WikiSQL and WikiTQ. We recall from Table 1 that overflow rate increases drastically with shorter length budgets. TaPEx and TaPas use different linearization and tokenization strategies, thus they yield different overflow rates for the same budget. However, for both models and datasets, 64 tokens lead to an 100% overflow rate. We show the DA values in WikiSQL and WikiTQ in Figure 2. In WikiSQL, ITR helps the TableQA model remain in the region of the best accuracy even when reducing the budget to 256 and 128 tokens for TaPEx and TaPas, respectively. Without the aid of ITR, both models sharply degrade in performance. Interestingly, TaPEx-based models are less robust when reducing the number of tokens than TaPas-based model. This is because TaPEx linearization uses up tokens faster, therefore encoding less table informa-

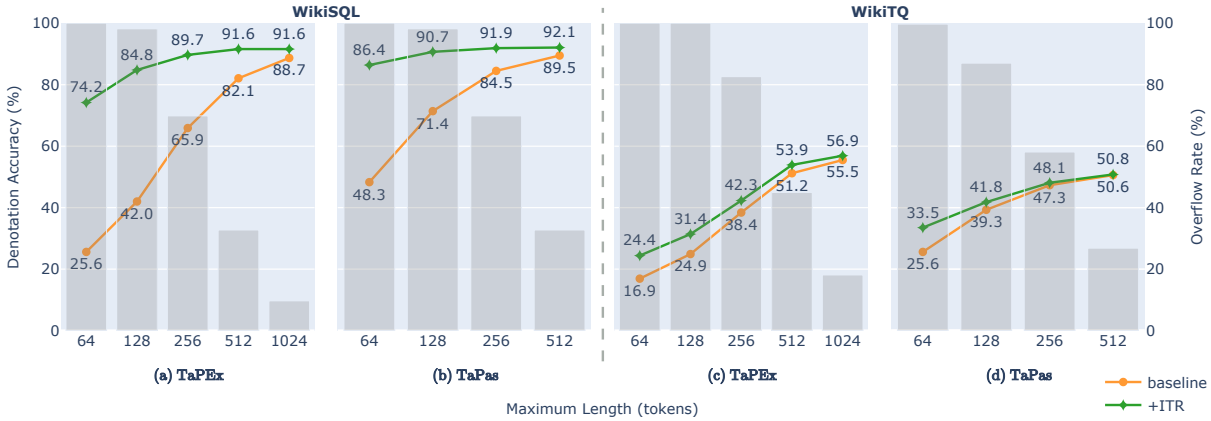

Figure 2: Impact of input length budget on Denotation Accuracy (line plots) and Overflow Rate (bar plots) for WikiSQL (left) and WikiTQ (right) Test sets.

tion overall (see Appendix D for a case study). In WikiTQ, while the trend remains unchanged, there is a more noticeable drop for all the models. This is due to the challenging nature of WikiTQ questions, which generally require visibility of larger portion of tables. However, ITR still improves over baselines, further widening the gap as we decrease the length budget to 128 and 64 tokens. Similarly to results in WikiSQL, TaPas is more robust than TaPEx in extreme scenarios in WikiTQ. Finally, while ITR benefits models in standard benchmarks, it is even more beneficial in extreme realistic scenarios.

6 Ablation Study

6.1 ITR Variants

In addition to our ITR, we investigate several variants: with varying number of sub-tables N, representing tables using columns or rows only, creating sub-tables with different strategies, or scoring items via a different measure of relevance.

Row/Column-only Items. Our ITR considers a mix of both row and column items. We redefine Algorithm 1 to consider only row or column items, but not both, creating the following ITR variants:

1. ITR$_{col}$: $\texttt{Items}(T)$ maps a table T into a set of columns, e.g., $\{c_1, c_2, c_3, c_4\}$ in Figure 1(a). A sub-table is created by combining the retrieved columns, e.g., Figure 1(b) would be represented as $\{c_1, c_3\}$ but containing all 4 rows for the 2 columns.
2. ITR$_{row}$: similar to ITR$_{col}$, but only considering row-wise items.

Reduction *versus* Addition. ITR returns the largest possible sub-tables by iteratively dropping irrelevant items to fully leverage the length budget (here referred to as *Reduction*, and models are suffixed by '$^-$'). As such, we do not consider the smallest sub-tables that only contain the top-N relevant items. To verify whether dropping the irrelevant items is a better approach, we contrast it with an *Addition* strategy (models suffixed by '$^+$'), where we return the top-N sub-tables created by successively appending the top-N items by their similarity with q, i.e., after Line 10 in Algorithm 1.

Semantic versus lexical. Finally, inspired by table input selection strategies in literature (Eisenschlos et al., 2020; Yin et al., 2020), we use n-gram similarity instead of dense retrieval in Algorithm 1 (Lines 5-6) and obtain ITR$_{ngram}$. We use ITR$_{ngram}$ to assess the importance of dense retrieval for ITR and its benefits in TableQA.

6.2 Results

In Table 6 we compare the accuracy of ITR variants on WikiSQL and WikiTQ. In WikiSQL, ITR$_{row}^{\{+,-\}}$ variations (#4 & #6) perform better than the baseline (#1) and the column-wise counterparts. In WikiTQ we see the opposite: ITR$_{col}^{\{+,-\}}$ variations (#3 & #5) perform better than baseline (#1) and the row-wise counterparts. However, ITR (#2) demonstrates superior performance across the board by jointly ranking the most relevant rows and columns, which strikes the right balance between the preference for each dataset. Even with N=1, ITR significantly improves the TaPEx baseline. Indeed, N=10 delivers only a slight improvement over N=1, by 0.3-0.4% in Test set depending on the dataset. As such, N=1 is a more efficient solution for applications with limited computational resources. We

#	Models	WikiSQL				WikiTQ			
		Dev	Test	Compact	Overflow	Dev	Test	Compact	Overflow
1	TaPEx	88.4	87.7	90.8	58.2	58.7	57.8	62.8	35.3
2	ITR → TaPEx	**91.3**	**91.4**	**91.6**	**89.4**	**61.8**	**61.5**	**65.2**	**44.8**
	with N=1	91.0	91.0	**91.6**	85.5	61.4	61.2	**65.2**	43.2
3	ITR^{-}_{col} → TaPEx	89.8	89.5	91.3	72.9	60.9	60.8	64.7	43.4
4	ITR^{-}_{row} → TaPEx	91.1	91.1	91.4	87.8	60.3	59.9	63.8	42.1
5	ITR^{+}_{col} → TaPEx	89.9	89.6	91.2	74.4	60.2	58.8	62.1	43.6
6	ITR^{+}_{row} → TaPEx	90.3	90.5	91.5	80.9	50.9	49.8	53.3	33.7
7	ITR_{ngram} → TaPEx	89.5	89.1	91.5	66.6	57.8	57.2	62.4	33.5

Table 6: Ablation study of ITR and its variants. Compact and Overflow are subsets of the Test split. ITR is applied both in training and inference. **Bold** denotes the best accuracies for each dataset split.

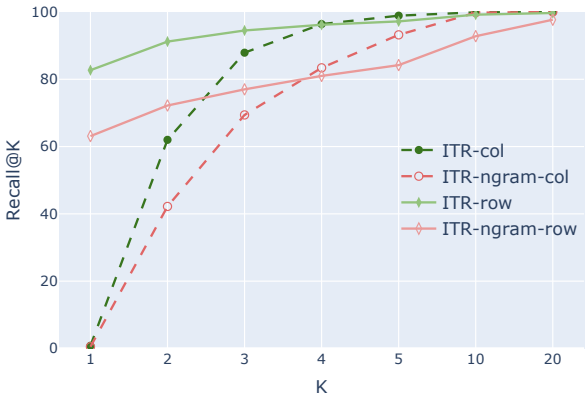

Figure 3: Results on the test set of WikiSQL for ITR and ITR_{ngram} item retrieval. For ITR_{ngram} we set $n \leq 3$, and plot $n = 3$ which shows always better performance.

further discuss varying N values in Appendix C.

Semantic versus lexical input selection. In Figure 3, we compare ITR and ITR_{ngram} when performing both row-wise, and column-wise retrieval. Neural-based ITR outperforms ITR_{ngram} for all values of K and both item types. The retrieval performance of ITR converges after $K > 5$ for ITR_{col} and $K > 10$ for ITR_{row}. For ITR_{ngram}, higher values of K are needed to achieve full Recall@K.

The lower performance of ITR_{ngram} is explained by poor lexical matching of the question with cell values and column names, as compared to the embedding similarities used by neural counterparts. For example, two viable questions that can be answered with the 'Rank" column in Figure 1 are "which mountain peak is the highest?" and the less natural "which is the mountain peak that has the lowest value in the rank?". While the latter matches with the column name, the former is more natural for human to ask and it does not match the column names via n-gram similarity. It is worth mentioning that WikiSQL is more canonicalized and ITR_{ngram} results in Figure 3 might be positively affected by the nature of WikiSQL. We expect the gap between ITR and ITR_{ngram} to be even larger in challenging datasets where the reference question is more human-like, e.g., WikiTQ. This is indeed evident when these variations are integrated in TableQA. Unsurprisingly, results in Table 6 (#7) show that, in contrast to ITR, ITR_{ngram} variation is able to improve the baseline performance (#1) in WikiSQL, but degrades the performance in WikiTQ.

7 Conclusions

In this paper we presented ITR for TableQA, an approach to create the most relevant sub-table(s) to efficiently answer a given question via TableQA. ITR is based on a dense retrieval component, which selects relevant rows and columns and combines them into a compact sub-table that satisfies length budget constraints. We combined ITR with different TableQA models from the literature, at training and/or inference time, and showed that ITR indeed captures the most relevant information, which enabled underlying models to perform better overall and become more robust, thus attaining state-of-the-art results on WikiSQL and WikiTQ benchmarks. ITR is flexible, does not depend on the underlying model and can be easily integrated in future model developments to increase their robustness. As future work we can combine ITR with computational operations over different table elements (Zhou et al., 2022) to collapse its information in a more compact format, to benefit also questions that rely on table completeness.

8 Limitations

First, in this work we limit the experimentation to vertical relational web-tables only, following the format of benchmarks used in TableQA, i.e., WikiSQL and WikiTQ. While we believe that ITR can easily be extended to horizontal entity web-tables, e.g., tables from Wikipedia, we do not expect our algorithm to transparently work on other types of tables that we do not consider, e.g., matrix tables from scientific papers and/or spreadsheets (Wang et al., 2021), where table items can be represented differently. However, this is not a limitation of the algorithm itself and adjusting our assumptions to certain scenarios and type of data can be feasible in the future. Second, ITR selects the relevant table elements by using a question as query. Therefore, it can only be applied for tasks with table-text joint input such as TableQA we showcase in the paper, or also table entailment tasks, e.g., table fact verification. Unfortunately, ITR cannot be used for tasks where table is the only input, e.g., table-to-text task. Finally, while ITR is beneficial for questions that do not rely on table completeness, its effectiveness is limited when, for example, all table cells are required to be predicted. Consider a question that requires cell counting, and the gold cells satisfying the query can be more than what we can feed a model with, e.g., "how many championship did Player A get?" and Player A has won 500 champions. However, this limitation does not arise from our approach and is rather inherited by existing TableQA models in the literature. Indeed, it can be a potential future direction of our work, which requires model innovation and table transformation that focuses on representing the information in a compact form.

References

Xinyun Chen, Petros Maniatis, Rishabh Singh, Charles Sutton, Hanjun Dai, Max Lin, and Denny Zhou. 2021. Spreadsheetcoder: Formula prediction from semi-structured context. In *International Conference on Machine Learning*.

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.

Haoyu Dong, Zhoujun Cheng, Xinyi He, Mengyu Zhou, Anda Zhou, Fan Zhou, Ao Liu, Shi Han, and Dongmei Zhang. 2022. Table pre-training: A survey on model architectures, pre-training objectives, and downstream tasks. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 5426–5435. International Joint Conferences on Artificial Intelligence Organization. Survey Track.

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

Julian Eisenschlos, Syrine Krichene, and Thomas Müller. 2020. Understanding tables with intermediate pre-training. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 281–296, 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.

Zhengbao Jiang, Yi Mao, Pengcheng He, Graham Neubig, and Weizhu Chen. 2022. OmniTab: Pretraining with natural and synthetic data for few-shot table-based question answering. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 932–942, Seattle, United States. Association for Computational Linguistics.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Syrine Krichene, Thomas Müller, and Julian Eisenschlos. 2021. DoT: An efficient double transformer for NLP tasks with tables. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3273–3283, Online. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*,

pages 7871–7880, Online. Association for Computational Linguistics.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2022. TAPEX: Table pre-training via learning a neural SQL executor. In *International Conference on Learning Representations*.

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.

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.

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 *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. 2021. Tuta: Tree-based transformers for generally structured table pre-training. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 1780–1790, New York, NY, USA. Association for Computing Machinery.

Jingfeng Yang, Aditya Gupta, Shyam Upadhyay, Luheng He, Rahul Goel, and Shachi Paul. 2022. TableFormer: Robust transformer modeling for table-text encoding. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 528–537, Dublin, Ireland. Association for Computational Linguistics.

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.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, bailin wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, richard socher, and Caiming Xiong. 2021. Gra{pp}a: Grammar-augmented pre-training for table semantic parsing. In *International Conference on Learning Representations*.

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

Fan Zhou, Mengkang Hu, Haoyu Dong, Zhoujun Cheng, Shi Han, and Dongmei Zhang. 2022. TaCube: Pre-computing Data Cubes for Answering Numerical-Reasoning Questions over Tabular Data. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

A Implementation Details

A.1 ITR Retriever Configuration

For the ITR retrieval component, i.e., to obtain question encoder (E_Q) and item encoder (E_T) in Algorithm 1, we fine-tune DPR on WikiSQL to retrieve the relevant table items for a question. We initialize E_Q and E_T using DPR weights released via `huggingface`, i.e., `facebook/dpr-question_encoder-single-nq-base` and `facebook/dpr-ctx_encoder-single-nq-base`, respectively. Before encoding via E_T, we linearize table items in a naive way with additional special tokens interleaving table cell values. For example, to encode row r_3 in Figure 1(a) we use: *<HEADER> rank <HEADER_SEP> mountain peak <HEADER_SEP> mountain range <HEADER_SEP> elevation <HEADER_END> <ROW> 2 <ROW_SEP> red slate mountain <ROW_SEP> sierra nevada <ROW_SEP> 13 , 149 ft <ROW_END>*.

We obtain annotations for *gold cells* by assessing the SQL query associated with each question-table-answer triple (q, T, a) in WikiSQL. For example, we can evaluate the following SQL query annotation "`SELECT Header1 FROM table WHERE Another Header = Some Entity`" to obtain cells that are selected as answers. In case of other aggregation functions beyond cell selection (e.g. COUNT, SUM, AVG), gold cells are those selected as input to aggregation functions. Thus, the table items containing any of these gold cells are gathered into positive items $I^+(q, T)$ while the remaining negative items are in $I^-(q, T)$.

In training ITR encoders, we leverage a contrastive loss to increase the output similarities between the question embeddings $E_Q(q)$ and the embeddings of positive items $E_T(i)$ ($i \in I^+(q, T)$). In contrast, the similarities with the embeddings of negative items are reduced. Formally, the embed-

Parameter	Value
Negative samples	4 (per positive sample)
Total GPUs	8
Learning rate	0.0001
Optimizer	Adam
Batch size	1 (per device)
Grad. accum. steps	4
Training steps	3,800 (ITR$_{col}$) 6,800 (ITR$_{row}$) 11,600 (ITR$_{mix}$)

Table 7: Best hyperparameters chosen for ITR retriever on the WikiSQL dataset.

dings of questions and items are:

$$\mathbf{e_q} = E_Q(q) \in \mathcal{R}^d; \mathbf{e_i} = E_T(i) \in \mathcal{R}^d, \quad (1)$$

where $i \in I \mid I = \texttt{Items}(T)$ and d is the hidden size. We use inner dot product as our similarity function:

$$r(q, i) = \mathbf{e_q}\mathbf{e_i}^\top \quad (2)$$

For each question, one positive item i^* and a few negative items are randomly sampled from $I^+(q, T)$ and $I^-(q, T)$ respectively. The training loss is therefore:

$$-\sum_{(q,I)} \log \frac{\exp\left(r(q, i^*)\right)}{\exp\left(r(q, i^*)\right) + \sum_{i \in I^-(q,T)} \exp\left(r(q, i)\right)} \quad (3)$$

In Table 7 we report other hyperparameters for the ITR retrieval component, chosen based on the Dev set of WikiSQL. We recall that we use the same checkpoint trained on WikiSQL also for WikiTQ in TableQA task. We train ITR retrieval component on an V100 machine, and the total training time for our main ITR variant is about 380 minutes.

A.2 Training with ITR Configuration

We use only TaPEx as a baseline for ITR at training time. We initialize TableQA models with the released checkpoint from `huggingface` for TaPEx pretraining, i.e., `microsoft/tapex-large`. As previously shown in Table 6, we notice a slight difference on the performance of the released TaPEx checkpoints via `huggingface` and the in-house fine-tuned TaPEx. Due to this, we report the hyperparameters we use to fine-tune TaPEx on WikiSQL and WikiTQ datasets in Table 8. We choose the best hyperparameters based on the performance on Dev set of each benchmark.

Parameter	Value (WikiSQL)	Value (WikiTQ)
Warmup steps	1000	0
Epochs	10	40
Learning rate	0.00003	0.00002
LR decay	Linear	
Optimizer	AdamW	
Total GPUs	8	
Batch size	1 (per device)	
Grad. accum. steps	4	
Weight decay	0.01	
Label smoothing	0.1	

Table 8: Best hyperparameters chosen for the in-house and ITR-enhanced TaPEx for WikiSQL and WikiTQ datasets.

A.3 Inference with ITR Models

In Table 9 we report the model checkpoints from `huggingface` that we used as baseline when applying ITR at inference time only. As mentioned in § 5, there are some differences between the performance obtained when evaluating the `huggingface` implementation of the baselines and the performance reported in each separate paper, mainly due to data processing and evaluation scripts. For example, OmniTab official repository was firstly based on that of TaBERT, which is based on encoder-only architecture. The authors have adjusted the code to an encoder-decoder architecture, however maintaining the tokenizer of TaBERT rather than TaPEx.This detail has not been transferred to the `huggingface` implementation. In addition, after the release of the original TaPas (Herzig et al., 2020), authors have implemented different variants on the same repository, including the preprocessing of the data and/or evaluation scripts. For example, Herzig et al. (2020) report that they drop examples if certain conditions are not satisfied, such as there is no scalar answer and the denotation cannot be found in the table. It is not clear if this decision continues to be true for the subsequent developments. Therefore, this does not allow a straightforward assessment of ITR contribution. To this end, we unify the implementations in a single evaluation framework, using the dataset splits, checkpoints and evaluation methods made available in the `huggingface` library for all the baselines. We release our unified framework upon acceptance.

Baseline	Checkpoints
	WikiSQL
TaPEx	`microsoft/tapex-large-finetuned-wikisql`
TaPas	`google/tapas-large-finetuned-wikisql-supervised`
	WikiTQ
TaPEx	`microsoft/tapex-large-finetuned-wtq`
TaPas	`google/tapas-large-finetuned-wtq`
OmniTab	`neulab/omnitab-large-finetuned-wtq`

Table 9: Checkpoints released via the `huggingface` library for TaPEx, TaPas and OmniTab, that we use as baselines for inference only experiments with ITR.

Training & Decoding Approach	Training Speed ↑ (iter/sec)	Training Batch Size	Training Time (mins)	Inference Speed ↑ (iter/sec)	Inference Batch Size
TaPEx	3.58	1	460	1.02	16
ITR → TaPEx	3.32	1	480	1.30	4

Table 10: Training and inference speed for TaPEx and ITR-enhanced TaPEx. We train each model on an A100 machine. Batch size is shown per GPU.

A.4 Computational Cost

In Table 10 we report training and inference speed for TaPEx and ITR-enhanced TaPEx. Especially in mix and row-wise ITR variants, the number of sub-tables is generally large (>100), which causes a significant performance bottleneck when dynamically tokenizing the sub-tables within the training/evaluation loop. There are two solutions regarding this problem. First, we can calculate the sub-tables at a preprocessing step which does not have any impact on the end-to-end training/inference speed. This is possible as choosing sub-table is not affected by the training updates. Second, to train and evaluate end-to-end, we leverage binary search to locate the valid sub-tables, i.e., sub-table(s) that first overflow, and stop processing the subsequent sub-tables which are guaranteed to be overflow. This allows to speed up the training by 3 times.

B Column and Row Order Effect

Despite the order of items returned by ITR, after creating and choosing the sub-tables, we rearrange their columns and rows in the same order as that in the original table. We rely on the order of the original training data, which can have its own biases in data creation. In addition, we observe that:

1. Exposing the most relevant items first at training time, in which case we also have access to gold items, leads to quick model over-fitting. The model can be strongly biased to choose cells that appear early in the linearized sequence, which is not desired for training a generalizable and robust TableQA model.

2. Baseline models have been trained without a strong bias on the column/row order, i.e., not enforcing that most relevant items are shown first. We show several experiments in which we apply ITR at inference time only. As such, introducing an ordering bias at inference time only decreases the performance.

Furthermore, to investigate whether the positioning of gold answers in the dataset can bias the trained model, we shuffle sub-table rows and columns to make the gold answers appear equally possible in any position of the input table. Results in Table 5) showed that shuffling at training time slightly increases the robustness of the model by 0.2-0.5 denotation accuracy points in WikiSQL Test and Dev sets respectively. Interestingly, shuffling has a bigger impact in the extreme scenario (see § 5.2) increasing the Overflow$_{ext.}$ by 2 denotation accuracy points. In the literature different strategies have been employed in the model design to avoid positional biases. For example, TableFormer (Yang et al., 2022) disposed of positional embedding to make all token positions homogeneous. However, such modifications of the baselines are out of the scope of this paper, in which we show the contribution of ITR on the current settings of each baseline.

N	WikiSQL Dev	WikiSQL Test
20	91.24	91.34
15	91.22	91.30
10	91.25	91.35
5	91.23	91.30
4	91.18	91.27
3	91.14	91.21
2	91.08	91.11
1	91.03	90.97
0	88.4	87.7

Table 11: DA of ITR → TaPEx for varying values of N. Underlined values denote the performance at our chosen N for the best model. N=0 indicates the baseline, i.e., using the full table.

C Multiple Sub-tables Effect

For our main experiments we use $N > 1$ sub-tables at inference time for generation baseline systems, i.e., TaPEx and OmniTab. In particular, we use $N = 10$ for our main ITR variant and ITR$_{ngram}$, while for the column and row only ITR variants, we set $N = 5$ and $N = 10$. In §5 (and Figure 3), we showed that ITR retrieval performance converges after $K > 5$ for columns and $K > 10$ for rows, which justifies the values selected in TableQA for N for each variant.

In §6 we showed the marginal impact of N=1 *versus* N=10. For completeness, in Table 11 we report the effect of varying N sub-tables for ITR → TaPEx: on the WikiSQL Test set, we get an improvement of 0.4 accuracy points for querying TaPEx on N=10 sub-tables *versus* doing so only on N=1 sub-table. Increasing N up to 20 yields no further improvements. We realize that using a large enough number of sub-tables, one might consider even simpler methods that consider different regions and combinations of the table each time, delegating the selection of the most relevant sub-table to the TableQA system, as per prediction confidence. For this reason, we also compare a naive baseline that uses up to N=10 randomly chopped sub-tables from a given table, without a specific notion of item relevance, combined with TaPEx. For this baseline, we simply sample columns and rows and combine them similar to ITR *mix* until a sub-table exceeds the token budget. Results show that N=10 random sub-tables might allow TaPEx to improve its performance by +2.3% in WikiSQL (*versus* +3.7% improvement from ITR). In WikiTQ, randomly choosing the sub-tables degrades the performance by -3.4% (*versus* +3.7% improvement from ITR). This is because in WikiSQL questions require less interaction between rows/columns and it might be enough for the system to have visibility of the items containing the gold answer. In WikiTQ, questions require aggregations between different columns and rows, and therefore a random combination of them leads to performance degradation.

We recall that for TaPas, we use N=1 as, due to joint tasks of cell selection and aggregation classification, it is not straightforward to determine the probability of the output making it unfeasible to compare $N > 1$ predictions.

D Case Study

In this Section we discuss two case studies. We illustrate the relevance assigned by ITR on the original table rows and columns as a heat-map where the color scale reflects the relevance scores per each column and row with respect to the question (green → yellow → orange → red). To obtain cell scores in their intersection, we sum up their corresponding column-wise and row-wise scores. As a result, more relevant cells are more red.

In Table 12 we show a side-by-side comparison of TaPas and TaPEx under the 64 token reduction scenario, and the benefit of applying ITR. TaPEx uses special tokens for encoding the table structure, which make the linearized sequence longer. TaPas instead, encodes the table structure via additional embedding layers. In addition, TaPas applies cell truncation to the first token for each cell, which are reconstructed as a post-processing step, and drops rows that exceed the token budget. This allows TaPas to be fed a larger portion of the table in the input, even if the cell information might be lost, e.g., in Figure 4 'OF-8' is squeezed into only 'OF' removing the distinction between 'Equivalent NATO Rank' across rows. As such, under extreme scenarios TaPas performs better than TaPEx, due to the visibility of a larger portion of the table. ITR proves beneficial and enables TaPEx to view only the relevant information within the token budget (Figure 6, left) to correctly answer the question.

In Table 13 we show a comparison of table pruning strategies included in TaPas, such as cell and row truncation, which might cause information loss. We disable those when we apply ITR, and provide TaPas with the full information contained on the question-relevant cells, as determined by ITR. In Figure 6 'New York' and 'New England Patriots'

Question: What could a Spanish Coronel be addressed as in the commonwealth military?
Gold Answer: Group Captain.

	Equivalent NATO Rank	Rank in Spanish	Rank in English	Commonwealth equivalent	US Air Force equivalent
0	OF-8	General del Aire	Lieutenant General	Air Marshal	Lieutenant General
1	OF-7	Brigadier General	Major General	Air Vice-Marshal	Major General
2	OF-5	Coronel	Colonel	Group Captain	Colonel
3	OF-4	Teniente Coronel	Lieutenant Colonel	Wing Commander	Lieutenant Colonel
4	OF-3	Mayor	Major	Squadron Leader	Major
5	OF-2	Capitán	Captain	Flight Lieutenant	Captain
6	OF-1	Teniente Primero	First Lieutenant	Flying Officer	First Lieutenant
7	OF-1	Teniente Segundo	Second Lieutenant	Pilot Officer	Second Lieutenant

Figure 4: Original Table from WikiSQL with ITR relevance heat-map.

Model	Input [question, table (serialized and tokenized)]	Prediction	Notes
TaPEx	<s> what could a spanish coronel be addressed as in the commonwealth military? col : equivalent nato rank code | rank in spanish | rank in english | commonwealth equivalent | us air force equivalent row 1 : of-8 | general del aire | lieutenant general | air marshal | lieutenant general</s> *(truncated at 64 tokens)*	Air Marshal	TaPEx uses separation tokens interleaving cell values.
TaPas	[CLS] what could a spanish coronel be addressed as in the commonwealth military? [SEP] equivalent rank rank commonwealth us of general lieutenant air lieutenant of brigadier major air major of coronel colonel group colonel of teniente lieutenant wing lieutenant of mayor major squadron major of capitan captain flight captain of teniente first flying first *(62 tokens, with only 1 token in each cell and truncated at 7 rows)*	Group Captain	TaPas uses additional embedding layers to encode table structure. The TaPas tokenizer by default squeezes the number of tokens in each cell to fit the table (e.g., 'OF-8' is squeezed into only OF), during which process information may be lost.

	Rank in Spanish	Rank in English	Commonwealth equivalent
2	Coronel	Colonel	Group Captain
5	Capitán	Captain	Flight Lieutenant

	Rank in Spanish	Rank in English	Commonwealth equivalent	US Air Force equivalent
2	Coronel	Colonel	Group Captain	Colonel
3	Teniente Coronel	Lieutenant Colonel	Wing Commander	Lieutenant Colonel
5	Capitán	Captain	Flight Lieutenant	Captain

Figure 5: Largest sub-table obtained for TaPEx (left) and TaPas (right) with ITR relevance heat-map. The largest sub-table for TaPas is bigger than that of TaPEx as the sequence length is calculated based on the tokenization of each TableQA model.

Model	Input [question, sub-table (serialized and tokenized)]	Prediction	Notes
ITR → TaPEx	<s> what could a spanish coronel be addressed as in the commonwealth military? col : rank in spanish | rank in english | commonwealth equivalent row 1 : coronel | colonel | group captain row 2 : capitán | captain | flight lieutenant</s> *(52 tokens without truncation)*	Group Captain	Now, the information being sought is within the length budget, leading to successful answering.
ITR → TaPas	[CLS] what could a spanish coronel be addressed as in the commonwealth military? [SEP] rank in spanish rank in english commonwealth equivalent us air force equivalent coronel colonel group captain colonel teniente coronel lieutenant colonel wing commander lieutenant colonel capitan captain flight lieutenant captain *(53 tokens without truncation)*	Group Captain	Now, the input sub-table is fully presented without harshly squeezing cell tokens.

Table 12: A case study with 64 token budget: comparing TaPEx and TaPas with or without ITR. ITR sub-table enables TaPEx to view the relevant information for correctly answering the question.

are both truncated as ‘New’ by TaPas. This information is crucial for correctly answering the question. Indeed, TaPas fails to locate the right cell after truncation, and predicts a wrong answer. The sub-table created by ITR in Figure 7 presents the full information of relevant cells to the model, thus enabling TaPas to make the correct prediction.

Question: Which winning team beat the New York Yankees?
Gold Answer: Arizona Diamondbacks.

	Year	Game or event	Date contested	League or governing body	Sport	Winning team	Losing team	Final score
0	2002	2001 World Series, game seven	November 4, 2001	Major League Baseball	Baseball	Arizona Diamondbacks	New York Yankees	3-2
1	2004	Super Bowl XXXVIII	February 1, 2004	National Football League	American football	New England Patriots	Carolina Panthers	32-29
2	2008	Super Bowl XLII	February 3, 2008	National Football League	American football	New York Giants	New England Patriots	17-14
3	2009	Super Bowl XLIII	February 1, 2009	National Football League	American football	Pittsburgh Steelers	Arizona Cardinals	27-23
4	2010	Winter Olympics men's hockey gold-medal game	February 28, 2010	International Olympic Committee	Ice hockey	Canada	United States	3-2 (overtime)
5	2011	NFL Week 15 game	December 19, 2010	National Football League	American football	Philadelphia Eagles	New York Giants	38–31

Figure 6: Original Table from WikiSQL with ITR relevance heat-map.

	League or governing body	Winning team	Losing team
0	Major League Baseball	Arizona Diamondbacks	New York Yankees
1	National Football League	New England Patriots	Carolina Panthers
2	National Football League	New York Giants	New England Patriots
4	International Olympic Committee	Canada	United States

Figure 7: Largest sub-table obtained for TaPas with ITR relevance heat-map.

Model	Input [question, sub-table (serialized and tokenized)]	Prediction	Notes
TaPas	[CLS] which winning team beat the new york yankees? [SEP] year game date league sport winning losing final 2002 2001 november major baseball arizona new 3 2004 super february national american new carolina 32 2008 super february national american new new 17 2009 super february national american pittsburgh arizona 27 2010 winter february international ice canada united 3 *(52 tokens, with only 1 token in each cell and truncated at 5 rows)*	New York Giants	TaPas tokenizer squeezes cell tokens in order to fit the table, which, however, confuses the model by having only one token "new" for both New York Yankees and New England Patriots. This prevents TaPas from finding the correct answer.
ITR → TaPas	[CLS] which winning team beat the new york yankees? [SEP] league or governing body winning team losing team major league baseball arizona diamondbacks new york yankees national football league new england patriots carolina panthers national football league new york giants new england patriots national football league philadelphia eagles new york giants *(53 tokens without truncation)*	Arizona Diamondbacks	ITR successfully presents most relevant information to TaPas. New York Yankees and New England Patriots are now fully presented, making question answering successful.

Table 13: A case study with 64 token budget: TaPas pruning strategies cause information loss, which confuses the model decision. ITR disables such information loss to remediate the previously wrong decision of TaPas.