Appendix

We provide details on our experimental setup and hyper-parameter tuning in Section A. Section B and C give additional information on model and the TABFACT dataset. We give details and results regarding our column pruning approach in Section D. Full results for SQA are displayed in Section E. Section F shows the accuracy on the pre-training tasks held-out sets. Section G contains the trigger words used for identifying the salient groups in the analysis section.

A Reproducibility

A.1 Hyper-Parameter Search

The hyper-parameters are optimized using a black box Bayesian optimizer similar to Google Vizier (Golovin et al., 2017) which looked at validation accuracy after $8,000$ steps only, in order to prevent over-fitting and use resources effectively. The ranges used were a learning rate from 10^{-6} to 3×10^{-4}, dropout probabilities from 0 to 0.2 and warm-up ratio from 0 to 0.05. We used 200 runs and kept the median values for the top 20 trials.

In order to show the impact of the number of trials in the expected validation results, we follow Henderson et al. (2018) and Dodge et al. (2019). Given that we used Bayesian optimization instead of random search, we applied the *bootstrap* method to estimate mean and variance of the max validation accuracy at $8,000$ steps for different number of trials. From trial 10 to 200 we noted an increase of 0.4% in accuracy and a standard deviation that decreases from 2% to 1.3%.

A.2 Hyper-Parameters

We use the same hyper-parameters for pre-training and fine-tuning. For pre-training, the input length is 256 and 512 for fine-tuning if not stated otherwise. We use $80,000$ training steps, a **learning rate of $\mathbf{2e^{-5}}$** and a **warm-up ratio of 0.05**. We disable the attention dropout in BERT but use a **hidden dropout probability of 0.07** . Finally, we use an Adam optimizer with weight decay with the same configuration as BERT.

For SQA we do not use any search algorithm and use the same model and the same hyper-parameters as the ones used in Herzig et al. (2020). The only difference is that we start the fine-tuning from a checkpoint trained on our intermediate pre-training entailment task.

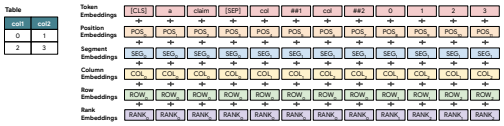

Figure 1: Input representation for model.

A.3 Number of Parameters

The number of parameters is the same as for BERT: $110M$ for base models and $340M$ for Large models.

A.4 Training Time

We train all our models on **Cloud TPUs V3**. The input length has a big impact on the processing speed of the batches and thus on the overall training time and training cost. For a BERT-Base model during training, we can process approximately 8700 examples per second at input length 128, 5100 at input length 256 and 2600 at input length 512. This corresponds to training times of approx. **78 minutes, 133 minutes and 262 minutes**, respectively.

A BERT-Large model processes approximately 800 examples per second at length 512 and takes **14 hours** to train.

B Model

For illustrative purposes, we include the input representation using the 6 types of embeddings, as depicted by Herzig et al. (2020).

C Dataset

Statistics of the TABFACT dataset can be found in table 1.

	Statements	**Tables**
Train	92,283	13,182
Val	12,792	1,696
Test	12,779	1,695
Total	118,275	16,573
Simple	50,244	9,189
Complex	68,031	7,392

Table 1: TABFACT dataset statistics.

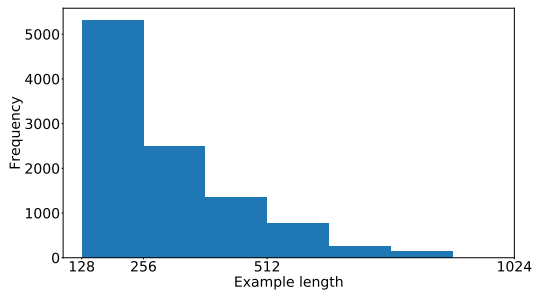

Figure 2: Input length histogram for TABFACT validation dataset when tokenized with BERT tokenizer.

D Columns selection algorithm

Let $cost(.) \in \mathbb{N}$ be the function that computes the number of tokens given a text using the BERT tokenizer, t_s the tokenized statement text, t_{c_i} the text of the column i. We denote the columns as $(c_1, .., c_n)$ ordered by their scores

$$\forall i \in [1, ..n-1] f(c_i) > f(c_{i+1})$$

where n is the number of columns. Let m be the maximum number of tokens. Then the cost of the column must verify the following condition.

$$\forall i \in [1..n], c_i \in C_{+_i} \text{if}$$
$$2 + cost(t_s) + \sum_{t_{c_j} \in C_{+_{i-1}}} cost(t_{c_j}) + cost(t_{c_i}) \leq m$$

where C_{+_i} is the set of retained columns at the iteration i. 2 is added to the condition as two special tokens are added to the input: $[CLS], t_s, [SEP], tc_1, ..., tc_n$. If a current column c_i doesn't respect the condition then the column is not selected. Whether or not the column is retained, the algorithm continues and verifies if the next column can fit. It follows C_{+_n} contains the maximum number of columns that can fit under m by respecting the columns scoring order.

There is a number of heuristic pruning approaches we have experimented with. Results are given in 2.

Word2Vec (W2V) uses a publicly available word2vec (Mikolov et al., 2013) model[1] to extract one embedding for each token. Let T_S be the set of tokens in the statement and T_C the set of tokens in a column. The cosine similarity for each pair is given by $\forall (s, c) \in T_S \times T_C$

$$f(s,c) = \begin{cases} 1 & \text{if s = c} \\ 0 & \text{if s or c are unknown} \\ \cos(v_s, v_c) & \text{else} \end{cases}$$

[1] https://tfhub.dev/google/tf2-preview/gnews-swivel-20dim/1

where v_i represents the embedding of the token i. For a given column token c we define the relevance with respect to the statement as the average similarity to every token:

$$f(S,c) = \text{avg}_{s \in T_S : f(s,c) > \tau} f(s,c)$$

Where τ is a threshold that helps to remove noise from unrelated word embeddings. We set τ to 0.89. We experimented with max and sum as other aggregation function but found the average to perform best. The final score between the statement S and the column C is given by

$$f(S,C) = \max_{c \in T_C} f(S,c)$$

Term frequencyâĂŞinverse document frequency (IWF) Scores the columns' tokens proportional to the word frequency in the statement and offset by the word frequency computed over all the tables and statements from the training set.

$$f(t_s, c) = \frac{TF(t_s, c)}{\log(WF(c) + 1)}$$

Where $TF(t_s, c)$ is how often the token c occurs in the statement t_s, and $WF(c)$ is the frequency of c in a word count list. The final score of a column C is given by

$$f(t_s, C) = \max_{c \in T_C} \left(\frac{TF(t_s, c)}{\log(WF(c) + 1)} \right)$$

Character N-gram (CHAR) Scores columns by character overlap with the statement. This method looks for sub-list of wordâĂŹs characters in the statement. The length of the characters' list has a minimum and maximum length allowed. In the experiments we use 5 and 20 as minimum and maximum length. Let $\mathbb{L}_{s,c}$ be the set of all the overlapping characters' lengths. The scoring for each column is given by

$$f(t_s, t_c) = \frac{min(max(\mathbb{L}_{s,c}, 5)), 20)}{cost(t_c)}$$

Method	PT Size	FT Size	Val
TABLE-BERT		512	66.1
OURS	512	512	78.3 ± 0.2
	256	512	78.6 ± 0.3
	128	512	77.5 ± 0.3
OURS - HEL	128	512	76.7 ± 0.4
	128	256	76.3 ± 0.1
	128	128	71.0 ± 0.3
OURS - HEM	256	512	78.8 ± 0.3
	256	256	78.1 ± 0.1
	128	512	78.2 ± 0.4
	128	256	77.0 ± 0.2
	128	128	72.7 ± 0.2
OURS- W2V	128	512	77.7 ± 0.3
	128	256	76.0 ± 0.2
	128	128	70.6 ± 0.3
OURS- IWF	128	512	77.9 ± 0.2
	128	256	77.2 ± 0.1
	128	128	72.7 ± 0.3
OURS- CHAR	128	512	77.5 ± 0.2
	128	256	74.8 ± 0.1
	128	128	68.7 ± 0.0

Table 2: Accuracy of different pruning methods: The heuristic entity linking (HEL) (Chen et al., 2020), Heuristic exact match (HEM), word-to-vec (W2V), inverse word frequency (IWF), character ngram (CHAR) at different pre-training (PT) and fine-tuning (FT) sizes. Error margins are estimated as half the interquartile range.

E SQA

Table 3 shows the accuracy on the first development fold and the test set. As for the main results, the error margins displayed are half the interquartile range over 9 runs, which is half the difference between the first and third quartile. This range contains half of the runs and provides a measure of robustness.

F Pre-Training Data

When training on the pre-training data we hold out approximately 1% of the data for testing how well the model is solving the pre-training task. In Table 4, we compare the test pre-training accuracy on synthetic and counterfactual data to models that are only trained on the statements to understand whether there is considerable bias in the data. Both datasets have some bias (i.e. the accuracy without table is higher than 50%.). Still there is a sufficient enough gap between training with and without tables so that the data is still useful.

The synthetic data can be solved almost perfectly whereas for the counterfactual data we only reach an accuracy of 84.3%. This is expected as there is no guarantee that the model has enough information to decide whether a statement is true or false for the counterfactual examples.

Data	Model	PT Size	Val_S	Val_C
Counterfactual	base	128		82.0
Counterfactual w/o table	base	128		76.0
Synthetic	base	128	94.3	
Synthetic w/o table	base	128	77.8	
Synthetic + Counterfactual	base	128	93.7	79.3
	base	256	98.0	83.9
	base	512	98.4	84.3
Synthetic + Counterfactual	large	128	94.3	81.0
	large	256	98.5	86.8
	large	512	98.9	87.3

Table 4: Accuracy on synthetic (Val_S) and counterfactual held-out sets (Val_C) of the pre-traininig data.

In table 5 we show the ablation results when removing the counterfactual statements that lack a supporting entity, that is a second entity that appears in both the table and sentence. This increases the probability that our generated negative pairs are incorrect, but it also discards 7 out of 8 examples, which ends up hurting the results.

Data	Val
Synthetic	77.6
Counterfactual	75.5
Counterfactual + Synthetic	78.6
Counterfactual (only supported)	73.6
Counterfactual (only supported) + Synthetic	77.1

Table 5: Comparisons of training on counterfactual data with and without statements that don't have support mentions.

G Salient Groups Definition

In table 6 we show the words that are used as markers to define each of the groups. We first identified manually the operations that were most often needed to solve the task and found relevant words linked with each group. The heuristic was validated by manually inspecting 50 samples from each group and observing higher than 90% accuracy.

Data	Size	ALL		SEQ		Q1		Q2		Q3	
		Dev	Test	Dev	Test	Dev	Test	Dev	Test	Dev	Test
MASK-LM	Base	60.0 ± 0.3	64.0 ± 0.2	35.3 ± 0.7	34.6 ± 0.0	72.4 ± 0.4	79.2 ± 0.6	59.7 ± 0.4	61.2 ± 0.4	50.5 ± 1.1	55.6 ± 0.7
Counterfactual	Base	63.2 ± 0.7	65.0 ± 0.5	39.3 ± 0.6	36.5 ± 0.6	74.7 ± 0.3	78.4 ± 0.4	63.8 ± 1.2	63.7 ± 0.3	52.4 ± 0.7	57.5 ± 0.7
Synthetic	Base	64.1 ± 0.4	67.4 ± 0.2	41.6 ± 0.8	39.8 ± 0.4	75.3 ± 0.7	79.3 ± 0.1	64.4 ± 0.6	66.2 ± 0.2	55.8 ± 0.7	60.2 ± 0.6
Counterfactual + Synthetic	Base	64.5 ± 0.2	67.9 ± 0.3	40.2 ± 0.4	40.5 ± 0.7	75.6 ± 0.3	79.3 ± 0.3	65.3 ± 0.6	67.0 ± 0.3	55.4 ± 0.5	61.1 ± 0.9
Counterfactual + Synthetic	Large	68.0 ± 0.2	71.0 ± 0.4	45.8 ± 0.3	44.8 ± 0.8	77.7 ± 0.6	80.9 ± 0.5	68.8 ± 0.4	70.6 ± 0.3	59.6 ± 0.5	64.0 ± 0.3

Table 3: SQA dev (first fold) and test results. ALL is the average question accuracy, SEQ the sequence accuracy, and QX, the accuracy of the X'th question in a sequence. We show the median over 9 trials, and errors are estimated with half the interquartile range .

Slice	Words
Aggregations	total, count, average, sum, amount, there, only
Superlatives	first, highest, best, newest, most, greatest, latest, biggest and their opposites
Comparatives	than, less, more, better, worse, higher, lower, shorter, same
Negations	not, any, none, no, never

Table 6: Trigger words for different groups.

References

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2020. Tabfact: A large-scale dataset for table-based fact verification. In *Proceedings of the International Conference on Learning Representations*, Addis Ababa, Ethiopia.

Jesse Dodge, Suchin Gururangan, Dallas Card, Roy Schwartz, and Noah A. Smith. 2019. Show your work: Improved reporting of experimental results. In *Proceedings of Empirical Methods in Natural Language Processing*, pages 2185–2194, Hong Kong, China. Association for Computational Linguistics.

Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Elliot Karro, and D. Sculley, editors. 2017. *Google Vizier: A Service for Black-Box Optimization*.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2018. Deep reinforcement learning that matters. In *Association for the Advancement of Artificial Intelligence*.

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 Association for Computational Linguistics*, pages 4320–4333, Online. Association for Computational Linguistics.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of Advances in Neural Information Processing Systems*, NIPSâĂŹ13, page 3111âĂŞ3119, Lake Tahoe, Nevada. Curran Associates Inc.