Testing the Limits of Unified Seq2Seq LLM Pretraining on Diverse Table Data Tasks

Soumajyoti Sarkar, Leonard Lausen
aws ai
{soumajs,lausen}@amazon.com

Abstract

Tables stored in databases and tables which are present in web pages and articles account for a large part of semi-structured data that is available on the internet. It motivates the need to develop a modeling approach with large language models (LLMs) which can be used to solve diverse table tasks such as semantic parsing, question answering as well as classification problems. Traditionally, there existed separate sequence to sequence models specialized for each table task individually. It raises the question of how far can we go to build a unified model that works well on some table tasks without significant degradation on others. To that end, we attempt at creating a shared modeling approach in the pretraining stage with encoder-decoder style LLMs that can cater to diverse tasks. We evaluate our approach that continually pretrains and finetunes different model families of T5 with data from tables and surrounding context, on these downstream tasks at different model scales. Through multiple ablation studies, we observe that our pretraining with self-supervised objectives can significantly boost the performance of the models on these tasks. Our work is the first attempt at studying the advantages of a unified approach to table specific pretraining when scaled from 770M to 11B sequence to sequence models while also comparing the instruction finetuned variants of the models.

1 Introduction

In recent years, there has been an emerging and quite substantial interest in specializing these LLMs [1, 2] on tasks with semi-structured data, for example knowledge graphs, spreadsheets, relational databases and tables which are of interest in this paper. Notably, there have been three streams of work that have been gaining traction in the realm of LLM pretraining for table specific tasks: **(1)** The first group of studies focus on predicting labels (essentially one column of a table) for classification and regression scenarios, using row values and column schema (from the other columns) as input [3–6]; [7–10]. They use gradient descent-based end-to-end learning with task-specific model pretraining and fine-tuning, **(2)** the second group of studies aim to specialize LLMs on table data to retrieve task-agnostic table/column/row representations for different downstream table understanding tasks. Drawing inspiration from models like BERT [11], these studies [12–15] serialize tables to a sequence of tokens and train them on textual self-supervised objectives, and **(3)** yet, there is a third group of studies that aim to use the generative nature of language models for data-to-text generation tasks where the nature of *data* entails some form of table and/or related text and the output *text* generally corresponds to text from the table or an output like SQL [16–19] (more details on these can be found in Section 1 of the Appendix).

Putting the above into the landscape of work that has been popular, the goal of our study is to conduct self-supervised pre-training with table specific data along with text related to the tables as input, followed by pre-finetuning for improving generalization to diverse tasks. While it is clear that there

Table Representation Learning Workshop at NeurIPS 2023

has been disparate studies with tabular data, some aiming at Question Answering (QA) tasks, some at getting row/column/table representations for semantic search and some at more complex reasoning tasks, what has not been done so far is to meaningfully conduct a methodological study of *large* scale pretraining with table data that can be *either* finetuned for specific downstream table tasks *or* help in scenarios where there is scarcity of data for a specific task. Specifically, the last group of studies in our above categorization focus on pretraining that can equip the models for end-end generation abilities suitable for table semantic parsing, table-based QA, table summarization among other tasks. However, pretraining with specialized objectives in the absence of large scale data and scale of models, remain key to transfer learning over diverse downstream tasks. We revisit several questions around table based LLM pretraining that have been studied before, albeit on a much smaller scale than the data used in this study. We attempt at answering the following:

RQ1: Can an intermediate pretraining stage with large scale data and multiple self-supervised text-text objectives transfer well to diverse downstream table tasks of both generation and encoding and can it improve performance on them compared to the text focused base models that we start with?

RQ2: Can this intermediate pretraining along with added pre-finetuning help in improving the generalization capabilities of the models in situations where the downstream tasks have scarce training data?

RQ3: Do we see improvements on these tasks from the table specific pretraining as we scale the model sizes while keeping the compute budget fixed in pretraining [20]?

The main contribution of our paper has been to revisit how sequence to sequence language models had been trained for these use cases in silos previously and then design a unified pretraining stage so that a single model can transfer well to these tasks. We combine a mixture of data composed of table only samples, tables along with natural language text and tables along with SQL data in the pretraining stage that gives it the ability to generalize well during finetuning. As mentioned in the work done in UnifiedSKG [21], the challenge lies in the heterogeneous nature of the inputs and the outputs of the tasks. To handle that challenge, we follow the paradigm used in UnifiedSKG (which only contains a single/multi-task finetuning stage), albeit we add a pretraining stage and propose *Uni*fied *Tab*ular *P*re*t*raining - **UniTabPT**. We follow the `pretraining` $\rightarrow$ `multi-task prefinetuning` $\rightarrow$ `finetuning` paradigm, however we run several ablation studies to show how our method generalizes better to downstream tasks than the paradigm in UnifiedSKG and previous task specific baselines.

2 UniTabPT Framework

UniTabPT uses encoder-decoder architectures, specifically the T5 architecture proposed in [22]. We slightly abuse the notation "pretraining" in our work to denote intermediate pretraining as we start with the publicly available text checkpoints of the T5 models and continually train them with self-supervised objectives on table data. There are two key points we want to highlight in our pretraining method: (1) first, we want the model to understand the semantics of SQL queries along with natural language text, but grounded with the tables they are associated with, and (2) we want the model to specialize on three kinds of inputs: tables only, tables along with text and tables along with SQL.

One of the challenges of table pretraining with heterogeneous inputs is the balance of the scale of data present for different input types. In our work, obtaining good SQL and NL questions for tables is a bottleneck. We linearize the table inputs to row major format and use three unsupervised objectives that allow the model to learn table specific properties. We hypothesize that it would help overcome the scarcity of data and allow the model to learn the inductive bias that can transfer to downstream tasks with related but different inputs. Our work is close to the objectives used in [16] although we introduce different generation objectives. As will be described in the following section, our dataset is a mix of table only data and table+(text/SQL) data.

2.1 Self-Supervised Pre-training Objectives

There are three elements of the table structure that we need for the denoising objectives: (1) the natural language text or the SQL that represents the query (we will often refer to this part as simply *text* in the context of the inputs), (2) the table headers and, (3) the table cells which contain the table values. We unify all the possible input types i.e. tables, tables+text and tables+SQL into the same

UniTabPT Encoder Input	UniTabPT Decoder Input	UniTabPT Decoder Output
<context> <text_NL> text half 1 <header> col name 1 | col name 2 | . . . <row> 0 val 1 | val 2 | . . . <row> 1 val 1 | val 2 | . . .	<NL_completion> <text_NL> text half 2	<text_NL> text half 2
<context> <text_SQL> sql half 1 <header> col name 1 | col name 2 | . . . <row> 0 val 1 | val 2 |. . . <row> 1 val 1 | val 2| . . .	<SQL_completion> <text_SQL> sql half 2	<text_SQL> sql half 2
<context><missing_context> <header> col name 1 | col name 2 | . . . <row> 0 val 1 | val 2 | . . . <row> 1 val 1 | val 2 | . . .	<NL_generation> <text_NL> text	<text_NL> text
<context> <text_NL> t1 <sentinel 1> t3 <header> col name 1 | <sentinel 2>| . . . <row> 0 val 1 | <sentinel 3> t6 | . . . <row> 1 val 1 | val 2 | . . .	<denoising> <sentinel 1> t2 <sentinel 2> t4 <sentinel 3> t5	<sentinel 1> t2 <sentinel 2> t4 <sentinel 3> t5

Table 1: Few examples of encoder decoder inputs and outputs to the model based on the objectives. Colored words denote special tokens. The text following the <context> section contains the natural language query/text or SQL (whichever applies). The <header> following that consists of the table headers and finally the table content is row-major linearized with the <row> tags. The first three rows describe completion and generation objectives. The last row gives an example of a denoising objective. Note that when the input is only tables, then the corresponding linearized input contains the value <context><missing_context> in the context section. All our inputs contain tables and the text/SQL part is optional for the datasets which contain only tables and no associated parallel text.

format described in Table 1 and use the following objectives. More details on the objectives are present in Section 2 of the Appendix.

2.1.1 Denoising

We use the following masking based text to text input/output objectives.
1. MLM on table cells: We use the denoising objectives [23, 12, 24, 25, 14, 26] on table cell content similar to T5 style masking that was adopted in the original T5 model pretraining with text only tokens.
2. MLM on NL/SQL text: We use the same masking as the table cell masking mentioned above. The only difference is the larger proportion of tokens we mask in this component of the input. Note that in both these masking techniques, we mask tokens and not entire cells.
3. MCP on table headers: We follow the work done in TaBert [23] and add Masked Column Prediction on the table headers. Unlike MLM, we mask all tokens belonging to a single header value.

2.1.2 Generation and Completion

We define another set of pre-training strategies inspired by ToTTo [18] in addition to denoising. For each table, we apply the following: for 50% of the examples in our pretraining dataset, we use the table as input and the corresponding text as output which we term *generation* (see row 3 in Table 1). For the rest 50%, we split the text into two halves, where the first half is concatenated alongside the serialized table data to be sent as input to UniTabPT and the second half is then used as the target for prediction. This is similar to Prefix LM and allows the model to reconstruct the context and forces the model to learn associations between the context and the table content. We term this *completion*(see rows 1 and 2 in Table 2). We apply these two objectives for both text and SQL inputs alongside the table.

2.2 Combining the Objectives

To combine the objectives in a multi-task fashion, we follow the T5 style multi-task learning and following the recent trend [27] in mixing objectives during pretraining. For each example in a batch, we toss a fair coin and choose the denoising objectives 60% of the time and the generation objective

rest of the time. When the denoising objective is selected for a particular example, we apply a few heuristics for the model to learn the table structure better considering the three components of the input. These could be described as such: (1) when the input is table only, we consider MLM on table cells and MCP for headers simultaneously with 50% probability. For the rest of the time, we only apply MCP with headers. The goal of the second step is for the model to recover the column names based on the table input and without any other corruption, (2) when the input is table and surrounding text or SQL, we apply MLM on table cells and text and MCP on headers simultaneously.

2.3 Adding Special Tokens to Inputs

Identifier tagging in inputs has been utilized for code pretraining tasks [2] and mostly in the form of special tokens in instruction finetuning where each task can be prompted with a special token [28]. Since our aim is to use the model for diverse downstream tasks, we use several special tokens which are marked by colors in the examples shown in Table 1. There are three kinds of special tokens used: (1) the separators which identify the parts of the table and text inputs and (2) the task identifiers like <NL_completion> which identify what the objective corresponds to and, (3) the type of the context/text, whether it is natural language text or SQL (details in Section 4 of Appendix).

3 Pretraining Data

One of the salient aspects of our work is that we attempt at combining table only inputs along with tables+text and tables+SQL inputs in a unified pretraining stage. We hypothesize that models develop table understanding through these cross correlations that can transfer well to diverse downstream tasks during finetuning. To that end, we collect the following datasets which have mostly been used in most of the previous studies on tabular model pretraining so far, yet studied individually across papers. Note that these datasets contain a mix of table+text (like metadata, captions, questions), table + SQL and table only examples. Since the amount of manually filtered and good quality annotated parallel table+text and table+SQL data is small compared to all the available data, we use some datasets where we collect the surrounding text like captions and page title of the page they were crawled from and use them as the parallel text. A detailed description of each dataset and statistics can be found in Section 3 of Appendix.

1. TAPAS: We use the data used in the TAPAS paper [12] to collect the data for Wikitables and Infobox.
2. Dresden Web Tables: The Dresden Web Tables Corpus [29] is a collection of about 125 million data tables extracted from the Common Crawl.
3. TAPEX: w use the dataset used in training the TAPEX model [19] and the data is a synthetic corpus of tabular data along with synthetic SQL.
4. ToTTo: We use the data The dataset used in the ToTTo paper [18] which is a collection of 121,000 English Wikipedia tables with corresponding natural language descriptions.
5. Tabert: The dataset used in training the tabert model [23] is a collection of 26 million web tables and their associated natural language context.
6. GAP: We use the data used in [17] which synthesizes the training data by collecting parallel sentences for the tables.

Aggregating all these data constituted roughly 50 billion tokens for one pass over the data. We performed a few data processing steps for each of the datasets, but we mainly followed the preprocessing mentioned in the study which produced the Tabert model [23], and in the interest of the paper length, we list the steps for table processing in Section 4 of Appendix.

4 Evaluating UniTabPT

To answer the three questions we laid out in the Introduction, we evaluate the pretrained model through single task finetuning on several table tasks of generation and classification. We apply UniTabPT pretraining on the public text checkpoints of 770M and the 3B versions of the T5 and the Flan-T5 models and we later show comparison of our models on the 11B model versions. All details of the pretraining configurations, hyper-parameters and the training resources used have been detailed in Section 5 of the Appendix. [1] for reproducibility purposes. In all of our experiments, we

[1] we will release the pretrained checkpoints of all model sizes soon under Apache 2.0 license

Model (Parameters)	WikiTQ	WikiSQL	Tabfact	SQA	CoSQL	Sparc	FetaQA
BART (400M)	27.93	81.62	79.04	-	-	-	27.11
TAPEX (400M)	57.41	87.93	83.13	-	-	-	25.23
REASTAP (400M)	57.93	87.16	83.28	56.41	41.23	58.91	26.41
T5-large (770M)	43.8	86.88	80.48	55.31	49.10	56.77	29.19
T5-3B (3B)	50.6	89.13	85.52	59.23	54.33	63.09	30.91
Flan-T5-Large (770M)	45.4	87.1	82.84	58.16	56.22	61.92	31.86
Flan-T5-XL (3B)	53.57	89.91	87.62	67.81	57.39	63.30	32.16
T5-large + UniTabPT	44.79	87.24	83.41	57.15	49.26	57.17	30.27
T5-3B + UniTabPT	53.76	89.26	86.83	65.72	55.16	**64.82**	32.14
Flan-T5-Large + UniTabPT	50.82	87.32	83.41	63.86	57.73	62.79	31.98
Flan-T5-XL + UniTabPT	**60.29**	**90.1**	**89.12**	**70.86**	**58.71**	64.71	**33.12**
Flan-T5-XL + UniTabPT + RF	57.96	89.92	88.71	69.27	58.49	63.81	33.08

Table 2: Results showing the performance of the models on the dev set of the structured knowledge grounded tasks. - denotes we did not run the models on these tasks as we mainly wanted to compare the BART and TAPEX models on specific tasks. Higher the better for these metrics. For the database tasks namely CoSQL and Sparc, we report the average over Exact Match and Execution metrics. For the rest of the results, we report the corresponding metrics as reported in Table 4 of UnifiedSKG [21] For WikiSQL, we report the results for the fully-supervised setting. Bold values denote the best performance in the single task finetuning setting.

use the same input format for the downstream finetuning as we did for the pretraining mentioned in Table 1. We only use the encoder input formats and the relevant special tokens during downstream task finetuning and do not use the special tokens for the decoder side inputs. One additional heuristic we use for specializing our model to multi-turn dialogues on tables is that for examples, where we had multiple queries associated with the tables or examples which had both query and metadata along with tables in the pretraining data, we aggregated them in the following format: `query || query 2/metadata | query 3/metadata | ... | table data` .... We show how this helps later in downstream tasks with multi-turn dialogues/QA on tables.

4.1 Structured Knowledge Grounded Generation Tasks

The goal of the work done in UnifiedSKG [21] was to demonstrate that a single unified text-text framework is able to achieve (near) state-of-the-art performance on all structured knowledge grounded tasks, using a single, general-purpose approach. As mentioned by the authors, their work did not focus on a pretraining approach, rather on a novel methodology to unify different knowledge grounded tasks with a structured text-text framework suitable for sequence to sequence models. We reuse their framework[2] for our evaluation purposes with single task finetuning. We use the following tasks for our paper: WikiSQL, WikiTQ, CoSQL, Sparc, FetaQA, SQA and TabFact. We refer the audience to the paper [21] for more details on these tasks and Section 7 of the Appendix for details on these finetuning tasks. We use the train splits of the datasets as used in this framework [3] for training and the dev set for the evaluation metrics. We do not use the test sets to report the results.

Among these tasks, Sparc and CoSQL are database related tasks, and database schema can widely differ from the schema of tables, so we want to check whether the knowledge gained from table related data can transfer to these domains. We choose these seven tasks among all the tasks mentioned in UnifiedSKG, since we do not want to evaluate our model on tasks that use knowledge graphs or triples like table + text + passage as inputs. For each task, we report the results using our own finetuning and hyper-parameter search, and in the process, we were not able to match all the metrics

[2] https://unifiedskg.com/introduction/#what-is-structured-knowledge-grounding
[3] https://github.com/HKUNLP/UnifiedSKG

exactly as reported in [21] most likely due to differences in hyper-parameter settings including the number of epochs to convergence.. We list all the hyper-parameters/configurations for finetuning UniTabPT and the baseline models on each task separately in Section 6 of the Appendix. We consider three baselines specialized on table data for our work: TAPEX which specializes on QA with tables [19], BART which is the base model TAPEX uses [30] and REASTAP [31] which is pretrained on synthetic data also starting with BART pretrained checkpoint. We use these baselines since REASTAP has shown to outperform other table pretrained models specialized for these tasks like TAPAS [12], GraPPa [32], Tableformer [24], TaBERT [23] and so these studies are subsumed for the comparisons. All these models are 400M parameter variants and at the time of our study, we did not find larger model sizes from these studies for our comparison. We also note an implicit challenge of finding one baseline model that could be used for all the above tasks to compare with our model.

For answering RQ1 with generation tasks, we first evaluate whether our pretraining helps in improving the performance of the models on the tasks individually. We report the single task finetuning results in Table 2. Note that in all the models without UniTabPT pretraining, we do not modify the input structure during finetuning compared to the setting used in UnifiedSKG and the tables are serialized in a row major format. We separate the comparisons in three phases:

Comparison with baselines: We observe that both TAPEX and REASTAP 400M model variants perform much better than even the vanilla 3B parameter T5 and Flan-T5 models on WikiTQ and WikiSQL tasks which shows that the nature of synthetic data used in these models with reasoning objectives is helpful. However, for other tasks especially, for multi-turn dialogue table tasks such as SQA and CoSQL, we find that REASTAP performs worse than the 770M and 3B Flan T5 variants. Furthermore, we also conclude that while models like TAPEX and REASTAP are specialized to some tasks due to the nature of the synthetic data they use for the pretraining, our models with UniTabPT can overcome that and show consistent performance gains for all tasks. As one data point, we observe that our Flan-T5+UniTabPT(770M) model is almost 40% better than REASTAP (400M) model on CoSQL while they are still comparable on WikiTQ (on which the baselines are specialized) and where they have slightly better performance than our model.

Comparing the T5 model families: We observe that the instruction finetuned versions of the T5 models, namely the Flan model families, perform better than their T5 base models on all tasks despite the fact that we do not use in-context learning with instructions, rather the full finetuning setting. We attribute this performance improvement to the fact that the instruction finetuning data in Flan models contain several QA and reasoning tasks. When we apply UniTabPT on top of the T5/Flan T5 models, we observe that the intermediate pretraining improves the performance of the models on all tasks. The gains are almost linear and consistent considering the performances of the baseline T5 and Flan T5 variants. We also observe that for tasks such as the WikiTQ and SQA, the Flan T5 large (770M) model along with UniTabPT is comparable to the T5 3B model for most tasks and in some cases, better. This goes to show that such pretraining can mitigate the latency overhead that comes with larger models with comparable performances. We conclude that continually pretraining on large scale data can still bring improvement to the already instruction-finetuned Flan models.

Comparisons on database and multi-turn dialogue tasks: As one of the questions we set out to explore, we find that for tasks like CoSQL and Sparc which have database schema as inputs, our pretraining paradigm improves the performance of these models despite that our pretraining did not account for database inputs specifically in the form of database schema inputs. When we compare the improvements we see in SQA which is a form of multi-turn QA on tables, we see that UniTabPT brings significant improvement and we attribute that to the form of multi-turn context we curate for some of our inputs during pretraining.

For answering RQ2, we attempt at improving the generalization performance of the model even beyond UniTabPT with multi-task pre-finetuning followed by task specific finetuning on few samples. The goal is to understand whether UniTabPT can enable the models for better cross task transfer with multi-task pre-finetuning and enable better performance in situations where table task data can be scarce. This would enable us to verify that our `pretraining` $\rightarrow$ `multi-task prefinetuning` paradigm can still be useful for direct application in table tasks where data can be scarce. In UniTabPT, we did not explore any data sources that can help the model learn the alignment between the SQL and the text for a table and which we believe can improve the results on table semantic parsing tasks. In light of these observations and the recent positive results shown from pre-finetuning [33] due to cross-task transfer learning, we curate a multi-task finetuning dataset using

	Flan-T5-XL + UniTabPT	Flan-T5-XL + UniTabPT + PFT
WikiTQ	60.29	**61.35**
WikiSQL	90.1	**92.13**
Tabfact	89.12	**78.64**
SQA	70.86	**60.95**
CoSQL	36.74	**40.13**
Sparc	51.27	**56.38**
FetaQA	14.21	**19.83**

Table 3: Results showing the performance in the low data-resource setting. The results are not reported on the single task finetuned setting but with the setting after the prefinetuning (PFT) stage. The results are evaluated in the low data resource setting (finetuned on 30% of training data) and the PFT stage did not include the exact data of these tasks.

the following: (1) dataset used in the TAPEX paper where the inputs to outputs are of the form (sql,table) $\rightarrow$ text samples, (2) training sets of the data from WikiTQ, WikiSQL, KVRET tasks in UnifiedSKG suite, (3) text $\rightarrow$ sql pairs from all the training datasets in UnifiedSKG suite and, (4) in order to avoid catastrophic forgetting of the data from the unsupervised stage [34], we replay a part of the training dataset from the generation objective.

Note that this is a form of multi-task learning where we control the proportion of datasets to mix from among the above datasets from (1) to (4) so as not to overfit to any particular task. We end up having one million samples in this stage combining all the respective portions. The detailed proportion of the datasets (1) to (4) used in this stage is listed in Section 8 of the Appendix. We also add the column types (identified using Spacy) to the linearized inputs in this stage. We finetune the `Flan-T5-Xl` and `Flan-T5-Xl+UniTabPT` models on this dataset for 200 epochs. For evaluation, we directly evaluate the prefinetuned (PFT) model on the dev sets of WikiSQL and WikiTQ without any further finetuning on them. For other tasks, we finetune them with only 30% of the available training data. From the results in Table 3, we observe that for all the tasks where we perform finetuning in a low-data resource scenario with only 30% of the training data, the degradation with PFT is much less when we apply UniTabPT on top of Flan-T5-XL.

	BERT Large	Tabert $(K=3)_{Large}$	Flan-T5 Large + UniTabPT Encoder
WikiTable-CTP	86.31	**87.67**	87.23
WikiTable-CRP	85.49	86.22	**86.92**
Viznet-CTP	90.26	91.59	**91.71**

Table 4: Results showing the performance of encoder models on table classification tasks. Results denote micro F1 scores.

	T5 11B	Flan-XXL	Flan-XXL+ UniTabPT
WikiTQ	56.32	57.89	**62.45**
WikiSQL	90.16	90.83	**90.86**
Tabfact	88.64	89.07	**90.71**
SQA	69.57	70.25	**71.23**
CoSQL	58.17	59.21	**59.26**
Sparc	65.81	67.19	**67.41**
FetaQA	32.95	33.91	**35.15**

Table 5: Results showing the performance when scaling the model sizes to 11B while reducing the number of tokens trained with.

One of the advantages of using the special tokens and serializing all task inputs to a unified format is that the model retains some of the properties it learnt about tables and parallel text and we run an experiment to measure the impact that having a unified format brings to downstream tasks. To that end, we finetune a model on the SKG tasks where we do not use the special tokens from our unified format in both the encoder and decoder inputs. Additionally, we assign different tokens to the missing values and separate the table components by ",". We name this format `RF`. The results of the model Flan-XL+UniTabPT+RF in table 2 show clearly that removing the serialization format results in degradation of the performance on the tasks and in many cases wipe out the advantage of UniTabPT.

4.2 Table Classification Tasks

We use the tasks proposed in [35] where they study the problem of annotating table columns i.e., predicting column types and the relationships between columns using information from the table exclusively and without any surrounding context. The study uses a multi-task learning framework and uses pre-trained language models for the prediction and we replace their pretrained encoders with our model while keeping the rest of the framework same. We use the problem description from that study to describe the prediction tasks: (Column type prediction, CTP) which annotates a given column from among predefined types and (Column relation prediction, CRP) which annotates the relation between a pair of columns from a pre-defined label set of relations. These are standard classification tasks with inputs and labels. Notably, both these tasks require extracting the representations of the tables before using them with additional layers for classifications.

We extract the encoder part of the *Flan-T5-large+UniTabPT* model and use the mean of the embeddings from the last layer to get an equivalent representation of the [CLS] token in BERT [36]. The goal of this experiment is to measure the effectiveness of our training when using the encoder for non-generative table specific tasks. To that end, we compare our encoder with the BERT large encoder and the TaBert encoder since they are roughly of equal model sizes. Note that the encoder part of our Flan-T5-large+UniTabPT model is roughly equal to these baseline encoder model sizes. We observe results on the WikiTable dataset and the Viznet dataset for column type (CTP) and column relation prediction (CRP) tasks [35]. We use the framework used in Doduo[4] while keeping the same hyper-parameters in each run. Table 4 shows the F1 results for the Wikitable tasks and Micro F1 for the Viznet task. Note that we use the same linearization format as used during pretraining,so as to take advatnage of the learned knowledge acquired during the tasks. We observe the following: for all the tasks, our encoder model performs better than BERT which confirms that a unified pretraining framework can be useful even for classification tasks. Our model lags behind Tabert on the Wikitable-CTP task and we hypothesize that since we did not use the column types during pre-training, it could be one reason although we do not see the same sensitivity for Viznet-CTP task.

4.3 Scaling Models to 11B

For answering RQ3, we compare the impact of the pretraining when scaling the model to 11B parameters. We train the model on roughly two-thirds of the number of epochs/tokens used to train the 770M and 3B models. Due to compute budget limitations, we perform the pretraining only on the Flan-T5-XXL (11B parameters) model and we evaluate three models on the downstream SKG tasks: the T5 11B model, the Flan-T5 11B model and our Flan-T5-XXL+UniTabPT. Table 5 shows that even at the 11B model size, we still see improvements coming from the table pretraining albeit with smaller margins. We observe the following: we see gains with our UniTabPT pretraining comparing Flan-XXL and our model across all the tasks, however, the gains are diminishing when comparing the 770M and the 3B model versions. This suggests that even with the limited data we have, we can still keep getting improvements with table specific pretraining at the scale of 11B model sizes. However, we do observe that the improvements for SQL tasks are marginally lower compared to non-SQL tasks, and we attribute that to the smaller SQL pretraining dataset. One way to mitigate that could be to upsample the SQL data in our pretraining mix which we leave as future explorations.

5 Related Work

LLMs have become one popular medium to solve table related tasks [18, 37, 15, 38]. Table semantic parsing and table QA have been solved with different styles of pretraining. Table pretraining tasks mainly aim at understanding the structure of tables and extending them to retrieve task-agnostic generalized representations of tables that can be used for different downstream table-based tasks [12, 15, 14, 25]. These studies use different deep neural network architectures mainly in the form of transformer encoders that in addition to taking the entire table information in the form of a serialized string,also add different learnable parameters to encode the structure of the tables. Table representation learning that focuses on joint text and table understanding is a field of research that partially overlaps with our work. Another branch of joint text and table understanding work focuses on text generation from tables[18, 39, 40, 16]. In contrast to these studies,our work centers on the importance of unified pretraining and how the scale of data and model impacts the performance.

[4]https://github.com/megagonlabs/doduo

References

[1] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[2] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859*, 2021.

[3] Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. Tabtransformer: Tabular data modeling using contextual embeddings. *arXiv preprint arXiv:2012.06678*, 2020.

[4] Sercan Ö Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 6679–6687, 2021.

[5] Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C Bayan Bruss, and Tom Goldstein. Saint: Improved neural networks for tabular data via row attention and contrastive pre-training. *arXiv preprint arXiv:2106.01342*, 2021.

[6] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. In *NeurIPS*, 2021.

[7] Leo Grinsztajn, Edouard Oyallon, and Gael Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. URL https://openreview.net/forum?id=Fp7__phQszn.

[8] Zifeng Wang and Jimeng Sun. Transtab: Learning transferable tabular transformers across tables. In *Advances in Neural Information Processing Systems*, 2022.

[9] Kounianhua Du, Weinan Zhang, Ruiwen Zhou, Yangkun Wang, Xilong Zhao, Jiarui Jin, Quan Gan, Zheng Zhang, and David Paul Wipf. Learning enhanced representations for tabular data via neighborhood propagation. In *NeurIPS 2022*, 2022.

[10] Witold Wydmański, Oleksii Bulenok, and Marek Śmieja. Hypertab: Hypernetwork approach for deep learning on small tabular datasets. *arXiv preprint arXiv:2304.03543*, 2023.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 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, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

[12] Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 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, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.398. URL https://aclanthology.org/2020.acl-main.398.

[13] Pengcheng Yin, Graham Neubig, Wen tau Yih, and Sebastian Riedel. TaBERT: Pretraining for joint understanding of textual and tabular data. In *Annual Conference of the Association for Computational Linguistics (ACL)*, July 2020.

[14] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. Turl: table understanding through representation learning. *Proceedings of the VLDB Endowment*, 14(3):307–319, 2020.

[15] Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. TABBIE: Pretrained representations of tabular data. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3446–3456, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.270. URL https://aclanthology.org/2021.naacl-main.270.

[16] Ewa Andrejczuk, Julian Martin Eisenschlos, Francesco Piccinno, Syrine Krichene, and Yasemin Altun. Table-to-text generation and pre-training with tabt5. *arXiv preprint arXiv:2210.09162*, 2022.

[17] Peng Shi, Patrick Ng, Feng Nan, Henghui Zhu, Jun Wang, Jiarong Jiang, Alexander Hanbo Li, Rishav Chakravarti, Donald Weidner, Bing Xiang, et al. Generation-focused table-based intermediate pre-training for free-form question answering. 2022.

[18] Ankur P Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. Totto: A controlled table-to-text generation dataset. *arXiv preprint arXiv:2004.14373*, 2020.

[19] Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. Tapex: Table pre-training via learning a neural sql executor. *arXiv preprint arXiv:2107.07653*, 2021.

[20] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[21] Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I Wang, et al. Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. *arXiv preprint arXiv:2201.05966*, 2022.

[22] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[23] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. Tabert: Pretraining for joint understanding of textual and tabular data. *arXiv preprint arXiv:2005.08314*, 2020.

[24] Jingfeng Yang, Aditya Gupta, Shyam Upadhyay, Luheng He, Rahul Goel, and Shachi Paul. 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, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.40. URL https://aclanthology.org/2022.acl-long.40.

[25] Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. Tuta: tree-based transformers for generally structured table pre-training. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1780–1790, 2021.

[26] Haoyu Dong, Zhoujun Cheng, Xinyi He, Mengyu Zhou, Anda Zhou, Fan Zhou, Ao Liu, Shi Han, and Dongmei Zhang. Table pretraining: A survey on model architectures, pretraining objectives, and downstream tasks. *arXiv preprint arXiv:2201.09745*, 2022.

[27] Yi Tay, Mostafa Dehghani, Vinh Q Tran, Xavier Garcia, Jason Wei, Xuezhi Wang, Hyung Won Chung, Dara Bahri, Tal Schuster, Steven Zheng, et al. Ul2: Unifying language learning paradigms. In *The Eleventh International Conference on Learning Representations*, 2023.

[28] Victor Sanh, Albert Webson, Colin Raffel, Stephen H Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, et al. Multitask prompted training enables zero-shot task generalization. *arXiv preprint arXiv:2110.08207*, 2021.

[29] Julian Eberius, Maik Thiele, Katrin Braunschweig, and Wolfgang Lehner. Top-k entity augmentation using consistent set covering. SSDBM '15, 2015. doi: 10.1145/2791347.2791353.

[30] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[31] Yilun Zhao, Linyong Nan, Zhenting Qi, Rui Zhang, and Dragomir Radev. Reastap: Injecting table reasoning skills during pre-training via synthetic reasoning examples. *arXiv preprint arXiv:2210.12374*, 2022.

[32] Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Richard Socher, and Caiming Xiong. Grappa: Grammar-augmented pre-training for table semantic parsing. *arXiv preprint arXiv:2009.13845*, 2020.

[33] Vamsi Aribandi, Yi Tay, Tal Schuster, Jinfeng Rao, Huaixiu Steven Zheng, Sanket Vaibhav Mehta, Honglei Zhuang, Vinh Q Tran, Dara Bahri, Jianmo Ni, et al. Ext5: Towards extreme multi-task scaling for transfer learning. *arXiv preprint arXiv:2111.10952*, 2021.

[34] Sanyuan Chen, Yutai Hou, Yiming Cui, Wanxiang Che, Ting Liu, and Xiangzhan Yu. Recall and learn: Fine-tuning deep pretrained language models with less forgetting. *arXiv preprint arXiv:2004.12651*, 2020.

[35] Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. *Annotating Columns with Pre-trained Language Models*. Association for Computing Machinery, 2022. ISBN 9781450392495. URL https://doi.org/10.1145/3514221.3517906.

[36] Jianmo Ni, Gustavo Hernández Ábrego, Noah Constant, Ji Ma, Keith B Hall, Daniel Cer, and Yinfei Yang. Sentence-t5: Scalable sentence encoders from pre-trained text-to-text models. *arXiv preprint arXiv:2108.08877*, 2021.

[37] Tongzhou Wang and Phillip Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *International Conference on Machine Learning*, pages 9929–9939. PMLR, 2020.

[38] Julian Martin Eisenschlos, Syrine Krichene, and Thomas Müller. Understanding tables with intermediate pre-training. *arXiv preprint arXiv:2010.00571*, 2020.

[39] Ori Yoran, Alon Talmor, and Jonathan Berant. Turning tables: Generating examples from semi-structured tables for endowing language models with reasoning skills. *arXiv preprint arXiv:2107.07261*, 2021.

[40] Fei Wang, Zhewei Xu, Pedro Szekely, and Muhao Chen. Robust (controlled) table-to-text generation with structure-aware equivariance learning. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5037–5048, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.371. URL https://aclanthology.org/2022.naacl-main.371.

[41] Yibo Sun, Duyu Tang, Nan Duan, Jianshu Ji, Guihong Cao, Xiaocheng Feng, Bing Qin, Ting Liu, and Ming Zhou. Semantic parsing with syntax-and table-aware sql generation. *arXiv preprint arXiv:1804.08338*, 2018.

[42] Xi Victoria Lin, Richard Socher, and Caiming Xiong. Bridging textual and tabular data for cross-domain text-to-sql semantic parsing. *arXiv preprint arXiv:2012.12627*, 2020.

[43] Wenhu Chen, Ming-Wei Chang, Eva Schlinger, William Wang, and William W Cohen. Open question answering over tables and text. *arXiv preprint arXiv:2010.10439*, 2020.

[44] Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée Miller. Semantics-aware dataset discovery from data lakes with contextualized column-based representation learning. *arXiv preprint arXiv:2210.01922*, 2022.

[45] Wenhu Chen, Jianshu Chen, Yu Su, Zhiyu Chen, and William Yang Wang. Logical natural language generation from open-domain tables. *arXiv preprint arXiv:2004.10404*, 2020.

[46] Michał Pietruszka, Michał Turski, Łukasz Borchmann, Tomasz Dwojak, Gabriela Pałka, Karolina Szyndler, Dawid Jurkiewicz, and Łukasz Garncarek. Stable: Table generation framework for encoder-decoder models. *arXiv preprint arXiv:2206.04045*, 2022.

[47] Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. *arXiv preprint arXiv:1508.00305*, 2015.

[48] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.

[49] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*, 2018.

[50] Ping Wang, Tian Shi, and Chandan K Reddy. Text-to-sql generation for question answering on electronic medical records. In *Proceedings of The Web Conference 2020*, pages 350–361, 2020.

[51] Moshe Hazoom, Vibhor Malik, and Ben Bogin. Text-to-sql in the wild: a naturally-occurring dataset based on stack exchange data. *arXiv preprint arXiv:2106.05006*, 2021.

[52] Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, et al. Sparc: Cross-domain semantic parsing in context. *arXiv preprint arXiv:1906.02285*, 2019.

[53] Tao Yu, Rui Zhang, He Yang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, et al. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. *arXiv preprint arXiv:1909.05378*, 2019.

[54] Mihail Eric and Christopher D Manning. Key-value retrieval networks for task-oriented dialogue. *arXiv preprint arXiv:1705.05414*, 2017.

[55] Linyong Nan, Chiachun Hsieh, Ziming Mao, Xi Victoria Lin, Neha Verma, Rui Zhang, Wojciech Kryściński, Hailey Schoelkopf, Riley Kong, Xiangru Tang, et al. Fetaqa: Free-form table question answering. *Transactions of the Association for Computational Linguistics*, 10:35–49, 2022.

[56] Mohit Iyyer, Wen-tau Yih, and Ming-Wei Chang. Search-based neural structured learning for sequential question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1821–1831, 2017.

A Section 1: Unified LLM Pretraining For Tables

We lay out some fundamental use-cases which primarily motivate the need to consolidate the pretraining phase of several tabular tasks. The fragmented nature of tabular data tasks can be roughly categorized into the following:

1. **Generating SQL-like expressions with natural language queries**: The user's query for a database is expressed in the form of natural language text and the goal of the model is to convert the query text into a formal SQL-like expression that can be executed on the database to output the final answer [41, 42].
2. **Question Answering on tables**: The user expresses a query as natural language text as before, but instead of an intermediate step to execute the query to fetch the result, the goal of the model is to directly fetch the answer from the table either through aggregate operations or single cell selections [12, 43].
3. **Fetching representations of table elements for classification**: The goal here is to jointly learn contextual representations for utterances and the structured schema of database tables [23] or in many cases extracting column or row representations for use in downstream tasks [44]. These models typically involve encoders and we later show how extract the representations for these cases. These representations can be extracted at different granular levels and are also used for semantic search for table retrieval [25].
4. **Data to Text Generation**: One of the other use-cases of table language models is text generation. It aims to study the problem of natural language generation from tables with logical inference as intermediate steps [45, 46].

Dataset	Category	Size after filtering	Avg. # rows / # columns
(TAPAS) Infobox/ Wikitables	(T,Text)	5.5M	3.6/7.4
Dresden Web Tables 2014	T	15M	18.3/5.5
TAPEX	(T, SQL)	3.1M	15.3/5.7
GAP	(T,Text)	2M	12.8/5.8
Web Tables 2012	(T,Text) / T	25M	15.4/5.3
ToTTo	(T,Text)	107K	30.3/6.6

Table 6: Statistics/properties of the pretraining dataset. *T* in the category denotes a table.

B Section 2: Pretraining objectives

B.1 Denoising

MLM on table cells: We use the regular denoising objectives on table cell content similar to T5 style masking that was adopted in the original T5 model pretraining with text only tokens. This is achieved by randomly masking out spans of tokens and training the model to predict a target sequence containing the missing or corrupted tokens in the input table. The target consists of all of the dropped-out spans of tokens, delimited by sentinel tokens. We replace 15% of table cell tokens in the input with a mask/sentinel token. This helps the model capture relationships between the neighbouring cells and the related text and table headers. We use a mean span length of 3 for the T5 style span corruption.

MLM on NL/SQL text: We use the same masking as the table cell masking mentioned above. The only difference is the amount of tokens we mask in this part which is 50%. We use the same mean span length of 3 for the T5 style span corruption. Note that in both these masking techniques, we mask tokens and not entire cells.

MCP on table headers: We follow the work done in TaBERT [23] and add MCP on the table headers. We mask 40% of the total number of table headers. Unlike MLM, we mask all tokens belonging to a single header value. Intuitively, MCP encourages the model to recover column information from its contexts and in the process learn the relation between table column values and its headers.

B.2 Generation and Completion

For each table, we use the natural text generation abilities of T5 sequence-sequence models and apply the following: for 50% of the examples in our pretraining dataset, we use the table as input and the corresponding text as output which we term *generation*. For the rest 50%, we split the text into two halves, where the first half is concatenated alongside the table data to be sent as input to UniTabPT and the second half is then used as the target for prediction in the seq2seq generation objective. This is similar to Prefix LM and allows the model to reconstruct the context and forces the model to learn associations between the context and the table content. We term this *completion*. We apply these two objectives for both text and SQL inputs corresponding to the table.

C Section 3: Pretraining Data

TAPAS: We use the data used in the TAPAS paper [12] to collect the data for Wikitables and Infobox. It contains 6.2M tables (3.3M of classInfobox3and 2.9M of class WikiTable). We also extract related captions and the title from the tables and use them as the corresponding parallel text.

Dresden Web Tables: The Dresden Web Tables Corpus [29] is a collection of about 125 million data tables extracted from the Common Crawl. DWTC is a large and diverse corpus,containing tables from a wide variety of domains,including,Business,Education,Government,Healthcare. For some Dresden tables,we only have the tables and for the rest,we use the captions and titles as the parallel text.

TAPEX: For training on table and SQL data,we use the dataset used in training the TAPEX model [19] and the data is a synthetic corpus of tabular data along with synthetic SQL. It was created by automatically synthesizing executable SQL queries and their execution outputs. The corpus contains over 3M examples,each of which is associated with a SQL query and its execution output. We do not use the execution output in the pretraining stage although we use that in the prefinetuning part with mixture of supervised datasets.

ToTTo: We use the data The dataset used in the ToTTo paper [18] which is a collection of 121,000 English Wikipedia tables with corresponding natural language descriptions. Each table in the dataset is associated with a natural language description which we use as parallel text.

TaBERT: The dataset used in training the TaBERT model [23] is a collection of 26 million web tables and their associated natural language context. The tables were extracted from the Common Crawl although the timeline of the pages of the common crawl used in the collection does not overlap with the Dresden web tables corpus.

GAP: We use the data used in [17] which synthesizes the training data by collecting parallel sentences for the tables.

Note that since the Dresden Web tables corpus and the TaBERT training corpus contain examples which are orders of magnitude larger than the other data sources,we downsample data from these two sources while keeping the rest of the data sources as they are. Final numbers are reported in Table 6.

D Section 4: Pretraining Data Processing

We did a few data processing steps for each of the datasets, but we mainly followed the preprocessing steps mentioned in the paper which produced the TaBERT model [23], and in the interest of the paper, we list only some steps for table processing. . Note that since our downstream tasks do not contain tables with large rows, we did not handle large table pretraining in this paper which we leave as future work. Instead we resorted to table truncation using the following steps:

- For table-text pairs, we consider 3-gram overlap between the natural language text and the table rows (by expanding them as a concatenated string) and keep the rows with the highest overlap. We take a maximum among of 40 rows from the overlapping set. If number of overlapping rows are less than 40, we randomly pick from the non-overlapping rows to fill the rest.
- For table only examples, we randomly take 40 rows.
- We truncate cells within tables including column names to maximum of 10 words. (words are for space separated tokens).
- We truncate the natural language text or SQL query to 40 words.

Apart from the above, we follow some standard steps like sanitizing cell text, sanitizing context, merging similar columns, removing columns with duplicate names, removing columns with invalid headers and removing rows in which all cells have the same text.

	Batch Size (Samples)	Dropout	Learning Rate	No. of epochs
770M	4096	0.1	5e-5	7
3B	4096	0.1	5e-5	7
11B	2048	0.15	1e-5	5

Table 7: Hyper-parameter settings in the table pretraining stage. They are kept the same for both the original T5 and the Flan T5 variants for the respective sizes.

E Section 5: Pretraining Details

For all our models, we use the T5 and Flan T5 architectures of the public models. We refer the reader to the Huggingface APIs we used to download the model during training [5] for the details. For training the models on the collected pretraining tabular dataset with self supervised objectives, we used the same hyper-parameters for all models including the Flan model families ranging from 770M to 11B and we only modify the dropout and the learning rates at the start of the training. We use an input sequence length of 1024 tokens for all our pretraining runs and we pad and truncate with the tokenizer accordingly. Since our objectives in this stage are self-supervised, the output sequence lengths of the models are determined based on the masking ratio. However, we follow the implementation used in [22] that first estimates the number of tokens to be masked starting from a pre-determined output sequence length which in our case is also set to 1024 and the masking ratio. Note that we do not pack multiple sequences into the same input but instead use padding tokens when the inputs do not fill 1024 tokens. For the 770M and 3B models, we set the learning rate to 1e-5 and the dropout to 0.1 as we find that these avoid early over-fitting when starting from the text pretrained checkpoints of the models we considered in this study. Table 7 lists some of the hyper-parameters used in the training. We reiterate that for all our training runs, we start with the publicly available text checkpoints of the T5 and Flan-T5 model families without changing the architectures. We use AdamW optimizer with β_1, β_2 and ϵ set to 0.9, 0.95 and 1e-8 respectively. We use a weight decay of 0.1 and gradient clipping of 1.0 and we use a warmup of 1000 steps for all our training runs. We use Deepspeed Zero Stage 2 for distributed training of the 770M and 3B model variants and we use Zero Stage 3 configuration for the 11B models. All our training configurations use bfloat16 as the precision. As mentioned in the paper, we add special tokens to the public tokenizers (for the respective models) so that they are not tokenized during the training procedure - `<missing_cell>`, `<missing_column>`, `<missing_context>`, for missing table cells, table headers and surrounding context and `<row>`, `<header>`, `_|`, `<text_NL>`, `<text_SQL>`, `<SQL_completion>`, `<NL_completion>` as identifier tags. We especially find that adding the special tokens to the decoder side inputs as shown in helps in faster convergence of the loss functions especially, when the SQL+table data is smaller conpared to other forms of data used in pretraining.

F Section 6: Finetuning Details

For the evaluation tasks for finetuning, we follow the setup used in UnifiedSKG [6] and directly use their code with some modifications to hyper-parameters to get our results. The discrepancies in the results in our paper and the metrics reported in [21] is mainly due to the number of epochs run and the hyper-parameter settings which we found to vary across the tasks for the best results (we could not find the exact settings from the Github repository or the paper). We especially find that the results are sensitive to the hyper-parameters and for purposes for reproducibility, we have listed the main hyper-parameters we tested using grid-search in Table 10. We serialize the inputs of all the tasks according to the format demonstrated in Table 1 of our main paper. For all our experiments on REASTAP ([31]) 770M and 3B models, we use Deepspeed Stage 2 with 8 A100 GPUs for the distributed training. For the 11B models, we use Deepspeed Stage 3 with 8 A100 GPUs for the training.

[5] The HF models for T5: `https://huggingface.co/docs/transformers/model_doc/t5` and Flan T5: `https://huggingface.co/docs/transformers/model_doc/flan-t5`

[6] https://github.com/HKUNLP/UnifiedSKG/tree/main

G Section 7: Finetuning Tasks

Task	Category Input → Output	Train Examples	Input Length (Tokens)
WikiTQ	(Question, T) → Answer	11321	1024
WikiSQL	(Text, T) → SQL	56355	1024
Sparc	(Multi-Turn, D) → SQL	12059	512
CoSQL	(Dialog D) → SQL	2164	512
SQA	(Multi-Turn, T) → Answer	12275	1024
FetaQA	(Question, T) → Free-form Answer	7326	512
Tabfact	(Statement, T) → Answer	1024	1024

Table 8: Statistics/properties of the finetuning SKG datasets. *T* and *D* in the category denotes a table and a database respectively.

The goal of the work done in [21] was to demonstrate that a single unified text-text framework is able to achieve (near) state-of-the-art performance on all structured knowledge grounded tasks, using a single, general-purpose approach. As mentioned by the authors, their work did not focus on a pretraining approach, rather on a novel methodology to unify different knowledge grounded tasks with a structured text-text framework. We reuse their knowledge grounding framework for our evaluation with the T5 models. Although their work focused on multiple techniques of finetuning like single task, multi-task, prefix LM among others, in our work, we focus on single task finetuning, as we are mainly interested in measuring the improvements that come from the pretraining stage on different tasks. We use the following tasks for our paper: WikiSQL, WikiTQ, CoSQL, Sparc, FetaQA, SQA and TabFact. We refer the audience to the paper [21] for more details on these tasks. We provide a brief summary of the kind of tasks in Table 8. We mention some key points about the datasets:

- We use the train splits of the datasets as used in this framework [7] for training and the dev set for the evaluation metrics. We do not use the test sets to report the results.
- For the database tasks namely CoSQL and Sparc, we report the average over Exact Match and Execution metrics. For the rest of the results, we report the corresponding metrics as reported in Table 4 of UnifiedSKG [21].
- For the WikiSQL task, we report the results for the fully-supervised setting.

As can be noted, among these tasks, Sparc and CoSQL are database related tasks, and database schema can widely differ from the schema of tables, so we want to check whether the knowledge gained from table related data can transfer to these domains.

[7] https://github.com/HKUNLP/UnifiedSKG

Dataset	Input → Output	Proportion of Dataset
TAPEX ([19]	(Text, Table) → SQL	100%
ToTTo [18]	Table → Text	100%
WikiTQ [47]	(Text, Table) → Text (Answer)	100%
WikiSQL [48]	(SQL, Table) → Text (Answer)	100%
Pretraining Data	(Text/SQL, Table) → Text	8%
Spider [49]	Text → SQL	100%
MIMICSQL [50]	Text → SQL	150%
SEDE [51]	Text → SQL	150%
Sparc [52]	Text → SQL	150%
CoSQL [53]	Text → SQL	150%
KVRET [54]	Text → SQL	150%
FetaQA [55]	Text → SQL	150%
SQA [56]	Text → SQL	150%

Table 9: Datasets used in the prefinetuning stage. Proportion of dataset indicates the proportion of data from the total available samples within that data. When proportion is greater than 100%, it means we upsample the data by repeating a portion of the dataset again. For datasets which are part of the evaluation tasks in our paper namely, WikiTQ, WikiSQL, Sparc, CoSQL, SQA and FetaQA, we use the portion of the training datasets only, for the partitions we have created prior to evaluation.

H Section 8: Prefinetuning data Mixtures

One of the critical bottleneck of any multi-task learning stage which utilizes data from different tasks is the right mixture of datasets [28] to use that allows for better generalization. To that end of ensuring that the prefinetuning stage enables better generalization in cases where downstream table tasks may have scarcity of training data, we utilized the proportions of the datasets detailed in Table 9. Note that all of these are used for sequence to sequence generation objective and without any masking where the inputs to output generation is the objective itself. So this is a supervised learning stage. A few key points to note about the dataset and the way we use it for evaluation of this stage in the main paper. Since we use the WikiSQL and the WikiTQ training splits of the datasets in its entirety and since they also are tasks in our evaluation suite, we perform zero shot evaluation after this stage is applied and report the results in the main paper. For other tasks like CoSQL, SQA and FetaQA which are also part of our evaluation suite, we only use the SQL input and the NL text output parts of these datasets in the prefinetuning data mixture. Note that these are all (Text, Table) → SQL datasets and we do not use the table portions. The main reason behind this is that we want the model to learn text and SQL alignment which is one of the characteristics that the model would not learn well in the pretraining stage. Following this, for the evaluations, we only use 30% of the trainining dataset of each task when we evaluate the prefinteuned models on these tasks, so there will be some overlap between the training dataset and the portion of the dataset used in the prefinetuning stage (although the exact input to output formats are different due to table exclusion). In all of these, there is no leakage of the dev sets of the tasks into the prefinetuning data mixtures.

	Task	Batch Size (Examples)	Input max length	Dropout	Learning Rate	Weight decay	No. of epochs	Beam Size
770M	WikiSQL	128	1024	0.1	1e-5	0.01	60	4
3B	WikiSQL	128	1024	0.1	5e-5	0.01	60	4
11B	WikiSQL	64	1024	0.1	5e-5	0.01	60	4
REASTAP	WikiSQL	128	1024	0.1	5e-5	0.01	60	4
770M	WikiTQ	128	1024	0.1	1e-5	0.01	150	4
3B	WikiTQ	128	1024	0.1	5e-5	0.01	150	4
11B	WikiTQ	64	1024	0.1	5e-5	0.01	150	4
REASTAP	WikiTQ	128	1024	0.1	5e-5	0.01	150	4
770M	Tabfact	128	1024	0.1	1e-5	0.1	50	4
3B	Tabfact	128	1024	0.1	5e-5	0.1	50	4
11B	Tabfact	64	1024	0.1	5e-5	0.1	50	4
REASTAP	Tabfact	128	1024	0.1	5e-5	0.1	50	4
770M	CoSQL	128	1024	0.01	1e-5	0.01	150	1
3B	CoSQL	128	1024	0.01	5e-5	0.01	150	1
11B	CoSQL	64	1024	0.01	5e-5	0.01	150	1
REASTAP	CoSQL	128	1024	0.01	5e-5	0.01	150	1
770M	SQA	128	1024	0.1	1e-5	0.01	50	4
3B	SQA	128	1024	0.1	5e-5	0.01	50	4
11B	SQA	64	1024	0.1	5e-5	0.01	50	4
REASTAP	SQA	128	1024	0.1	5e-5	0.01	50	4
770M	Sparc	128	1024	0.1	1e-5	0.01	150	1
3B	Sparc	128	1024	0.1	5e-5	0.01	150	1
11B	Sparc	64	1024	0.1	5e-5	0.01	150	1
REASTAP	Sparc	128	1024	0.1	5e-5	0.01	150	1
770M	FetaQA	128	512	0.1	1e-5	0.01	150	4
3B	FetaQA	128	512	0.1	5e-5	0.01	150	4
11B	FetaQA	64	512	0.1	5e-5	0.01	150	4
REASTAP	FetaQA	128	512	0.1	5e-5	0.01	150	4

Table 10: Hyper-parameter settings for the finetuning tasks. They are kept the same for both the original T5 and the Flan T5 variants for the respective sizes where applicable.