Multitask Pretraining with Structured Knowledge for Text-to-SQL Generation

Robert Giaquinto[*], **Dejiao Zhang**, **Benjamin Kleiner**, **Yang Li**
Ming Tan, **Parminder Bhatia**, **Ramesh Nallapati**, **Xiaofei Ma**

AWS AI Labs

{rgiaq,dejiaoz,kleinerb,ylizam,
mingtan,parmib,rnallapa,xiaofeim}@amazon.com

Abstract

Many machine learning-based low-code or no-code applications involve generating code that interacts with structured knowledge. For example, one of the most studied tasks in this area is generating SQL code from a natural language statement. Prior work shows that incorporating context information from the database schema, such as table and column names, is beneficial to model performance on this task. In this work we present a large pretraining dataset and strategy for learning representations of text, tables, and SQL code that leverages the entire context of the problem. Specifically, we build on existing encoder-decoder architecture by introducing a multitask pretraining framework that complements the unique attributes of our diverse pretraining data. Our work represents the first study on large-scale pretraining of encoder-decoder models for interacting with structured knowledge, and offers a new state-of-the-art foundation model in text-to-SQL generation. We validate our approach with experiments on two SQL tasks, showing improvement over existing methods, including a 1.7 and 2.2 percentage point improvement over prior state-of-the-arts on Spider and CoSQL.

1 Introduction

Tables, relational databases, and other forms of structured knowledge (SK) encompass a massive amount of data across a wide range of applications. Extracting insights held in such data often requires proficiency in query languages like SQL, making it only accessible to the minority of people with the technical skills. A natural language interface, however, would expand access to these information exponentially. Likewise, querying via natural language allows users quickly hone in on an answer to their particular question, rather than visually scanning dense tables where the majority of the information is irrelevant to the user. To that end, we explore pretraining techniques for large language models that focus on the challenging interplay between structured and unstructured knowledge, and target a variety of downstream text-to-SQL tasks.

Recently there have been significant advancements in learning representations for tables (Yin et al., 2020; Herzig et al., 2020; Eisenschlos et al., 2020; Liu et al., 2022; Wang et al., 2021c; Yu et al., 2021; Cheng et al., 2022; Dong et al., 2022), which advanced the state-of-the-art in a range of table-to-text tasks, like table question-answering (Nan et al., 2022; Chen et al., 2021), fact verification (Chen et al., 2020; Aly et al., 2021), data-to-text (Parikh et al., 2020; Nan et al., 2021), and semantic parsing (Yu et al., 2019b; Zhong et al., 2017). While better table understanding benefits a range of tasks, pretraining focused on text-to-SQL has thus far received less attention. Pretrained encoders, such as TaBERT and TAPAS (Yu et al., 2021; Yin et al., 2020; Herzig et al., 2020), show that pretraining BERT-style encoders (Devlin et al., 2019) on tables with mask language modeling (MLM) loss produces a strong foundation model that can be extended for text-to-SQL. GRAPPA includes small amount of synthetic SQL code in the pretraining data to more specifically target the text-to-SQL task (Yu et al., 2021). These encoder-only approaches are, however, restricted in their generative capabilities as they must be combined with an additional module that is carefully designed to generate valid SQL code (Zhong et al., 2017; Wang et al., 2021a).

Encoder-decoder architectures like T5 (Raffel et al., 2020), on the other hand, exhibit better performance on text-to-SQL to-date when constraining the decoder with rules that check for syntactic correctness (Scholak et al., 2021). However, the T5-based models with exceptional text-to-SQL performance (Xie et al., 2022; Scholak et al., 2021) have still only been pretrained on natural language (NL) — begging the question, can text-to-SQL encoder-decoders benefit from pretraining on structured in-

[*]Corresponding author.

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics
Volume 1: Long Papers, pages 11067–11083

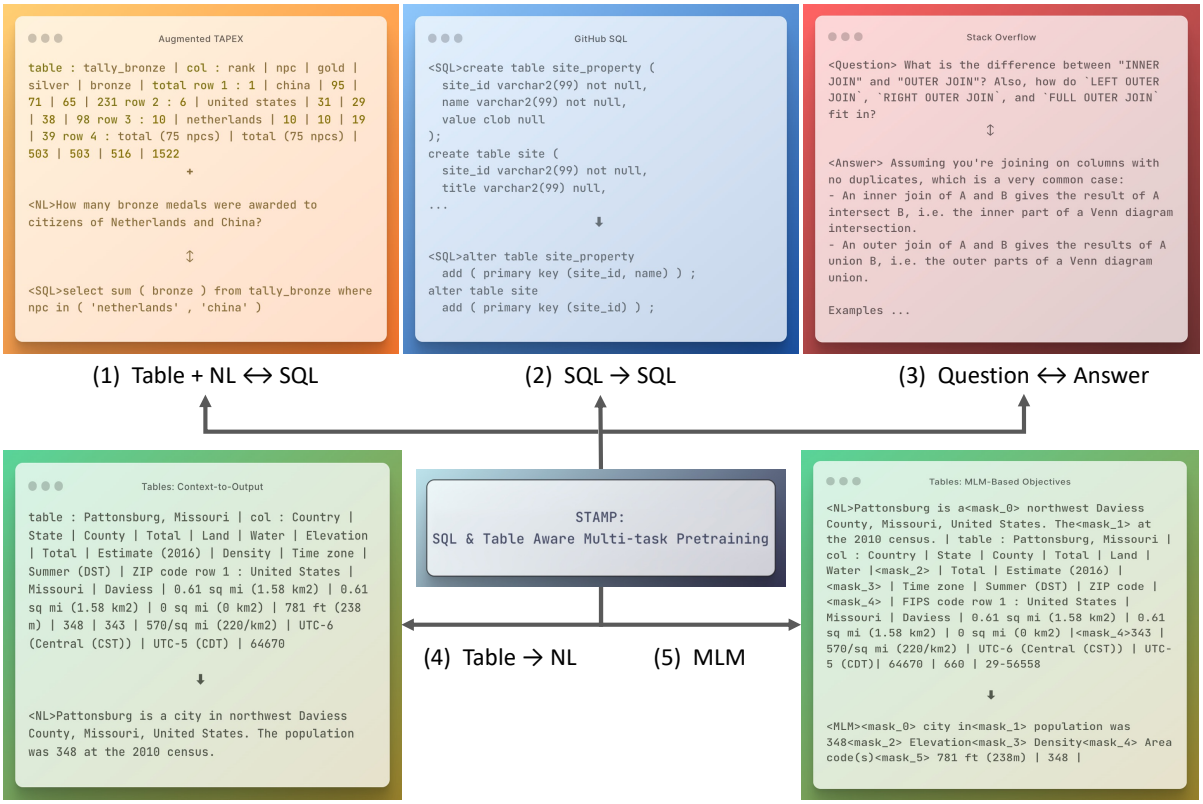

Figure 1: STAMP's multi-task strategy combines context-to-output with MLM-based objectives that are designed for our diverse of pretraining data. In (1-4) we show the context-to-output format for four data sources, however in pretraining MLM objectives are also applied. For example, (5) shows the MLM objectives applied to the same data as (4), showing the combination of the T5 style of masking with masked column recovery. For data sources (1) and (3) we also apply dual learning, where the context and output are interchanged to better align representations.

formation or code? Most recently, Andrejczuk et al. (2022) proposed a multi-task tabular pretraining strategy for T5 model, but their work introduced the tabular knowledge to the model with a single data source, i.e. Wikipedia tables.

In this work we introduce our *SQL and Table Aligned Multi-task Pretraining* (STAMP) framework, which explores pretraining encoder-decoder models for text-to-SQL. Starting from text-only T5 (Raffel et al., 2020) checkpoints, our multi-stage pretraining framework refines previous text-only models by continuing training on a collection of large multi-modal datasets that combine structured knowledge with natural language and SQL. Additionally, inspired by the impressive generalization of large language models incorporating code in pretraining data (Athiwaratkun et al., 2022; Brown et al., 2020; Chowdhery et al., 2022; Du et al., 2022; Thoppilan et al., 2022), we apply our pretraining framework to CodeT5 (Wang et al., 2021b) checkpoints that are trained on code.

Building on recent work in multi-task pretraining (Tay et al., 2022; Aghajanyan et al., 2021; Sanh et al., 2022; Aribandi et al., 2021), we combine masked language modeling (MLM) with task-aware context-to-output objectives that vary across tasks and datasets. For pretraining datasets with multiple modalities (i.e. combinations of NL, SQL, and structured knowledge) or intrinsic splits (e.g. question and answer), we explore the benefit of the dual learning objectives (Wang et al., 2021b). We assess our pretraining strategy on a variety of SQL benchmarks following the UnifiedSKG framework (Xie et al., 2022). Our approach outperforms previous text- and code-only pretraining, and gives a new state-of-the-art on a range of benchmarks. To better understand our strategy, we present ablation studies on the optimal objective mix, the impact of linearizing structured knowledge into row- versus column-centric tables, and the effect of building on previously pretrained text- versus code-only checkpoints. Our work shows that continued pretraining with multi-task learning is a promising direction for advancing the capacity of language models.

2 Related Work

Encoder-only Encoder-only transformer architectures like BERT and its successors (Devlin et al., 2019; Liu et al., 2019; Joshi et al., 2020; Reimers and Gurevych, 2019; Clark et al., 2020) optimize masked language modeling (MLM) objectives while using a bidirectional receptive field covering the whole input sequence. The encoder-only architectures perform well across a variety of tasks like classification, regression, sentiment analysis, question-answering, and retrieval. However, recent work (Herzig et al., 2020; Yin et al., 2020; Yu et al., 2021) shows that tasks like table-to-text and text-to-SQL require additional pretraining on structured knowledge for good generalization, and adapting MLM objectives to the unique structure of tabular data improves learning.

Prior to BERT, text-to-SQL models like SQL-Net and Seq2SQL (Zhong et al., 2017; Xu et al., 2017) encoded inputs with bidirectional LSTMs (Hochreiter and Schmidhuber, 1997) and generated queries via slot-filling. Text-to-SQL performance improved with the adoption of BERT-based encoders, for example (Yu et al., 2021; Wang et al., 2021a) attach feed forward networks and LSTMs to the BERT-style encoder to generate queries. Because encoder-only architectures are restricted in their ability to generate sequences, they require careful design to generate valid SQL queries and limit the complexity of those queries.

Encoder-Decoder Alternatively, encoder-decoders like BART (Lewis et al., 2019) and T5 (Raffel et al., 2020) combine the bidirectional encoder with a causal decoder are naturally suited for sequence-to-sequence tasks like text-to-SQL, and are quickly becaming the mainstream approach due to the reduced need for domain specific solutions (Qin et al., 2022). T5 (Raffel et al., 2020) in particular achieves impressive performance on a range of table-to-text and text-to-SQL tasks (Xie et al., 2022) despite pretraining that is limited to NL. Moreover, Shi et al. (2020) and Liu et al. (2022) leverage a BART-style encoder-decoder to improve the performance of pretrained models for text-to-SQL and table-to-text tasks, respectively. We follow this line, proposing a strategy that builds on top of T5 and CodeT5 (Wang et al., 2021b).

Multi-Task Training Raffel et al. (2020) explore various self-supervised objectives, and found the fill-in-the-blank style of denoising objective as most effective. Additionally, combining MLM objectives with small amounts of auxiliary objectives is effective (Liu et al., 2019; Aroca-Ouellette and Rudzicz, 2020). For encoder-decoder models Tay et al. (2022); Wang et al. (2021b) show the benefit of multi-task pretraining on a mix of the T5 span corruption objective (Raffel et al., 2020) along with a the causal language modeling (CLM) style of objective, similar to those used in decoder-only architectures (Brown et al., 2020). In the domain of text-to-SQL, Yu et al. (2021); Tao Yu et al. (2021) perform multitask learning by combining MLM with SQL specific objectives. Lastly, Xie et al. (2022); Aghajanyan et al. (2021); Aribandi et al. (2021); Sanh et al. (2022); FitzGerald et al. (2022); Chen et al. (2022) demonstrate that multi-task learning across a variety of datasets can improve performance relative to the single-task, single-dataset paradigm. Wang et al. (2021b) show that an objective mix specific to programming languages (PL) along with dual learning on bimodal data promotes generation on tasks combining PL and NL.

3 Multi-Task Pretraining on Structured Knowledge

Our *SQL and Table Aligned Multi-task Pretraining* (STAMP) model builds on the T5 encoder-decoder architecture and pretraining checkpoints (Raffel et al., 2020), and similarly our CodeSTAMP models build on the CodeT5 architecture and checkpoints Wang et al. (2021b). We develop a multi-task pretraining framework specifically designed to leverage our large and unique collection of data that combine various data modalities, namely natural language (NL), stuctured knowledge (SK), and SQL. STAMP introduces a new stage of pretraining that transitions T5 from being a purely NL programing language (PL) trained model to a backbone model that excels at text-to-SQL generation.

Next, we present the construction of our pretraining dataset in Section 3.1, the mixture of objectives designed to learn the unique structure of our data and align the NL, SK, SQL data modalities in Section 3.2, and our unified format for representing tasks and structured knowledge in Section 3.3.

3.1 Datasets and Pre-Processing

Our pretraining dataset consists of 18 million examples, with various combinations of NL, SQL code, and structured knowledge (see Figure 2). Our data is derived from diverse sources and we propose dif-

ferent strategies to remove many low-quality and noisy data from each data source. We tokenize the raw data using the corresponding T5 and CodeT5 tokenizers, which we augment to support new special tokens for representing input data modality, output tasks, and table structures. We process all data into sequences of up to 1024 tokens. More details on pre-processing are in Appendix A.

Table Data Approximately half of our pretraining data (N = 10,136,268) combine tables with NL. These table datasets derive from Wikipedia, WDC's Web Table Corpora, and arXiv. Pretraining on table datasets acts as a bridge from the previous text-only pretraining, while promoting alignment between NL and structured knowledge. In initial experiments we pretrained on all available table and NL pairs. However, after closer examination we discovered that a significant portion these examples exhibited minimal connection between the table and NL — and hence are unlikely to promote the desired alignment. Therefore, we choose to focus on high-quality examples and remove approximately 75% of the examples in which there is a tenuous or no connection between the table and the paired NL. To identify noisy examples we compute an edit similarity between the NL and the content of the table, we then drop examples with such similarity below a threshold. Likewise, to reduce noise within each example we truncate tables, keeping at most 6 rows and 25 columns which have the highest edit similarity between table and NL.

SQL Data The remainder of our pretraining data incorporate SQL. Approximately 10% of the examples (N = 1,918,468) are SQL code from GitHub repositories with permissive licenses. SQL code from GitHub only includes only a small amount NL in code comments, and some structured knowledge in the database schema definitions. We filter these data to remove duplicates and repetitive statements.

Approximately 25% of the examples (N = 4,479,767) are from SQL-related posts on Stack Overflow. These data combine NL questions and answers with snippets of SQL code, thereby bridging the NL knowledge learned during the prior text-only pretraining into domain-specific language, and aligning SQL with NL. We perform augmentations to increase the number of question-answer pairs and leverage hidden human supervision [1] in the data. We first create five augmented versions of each question using random word deletion, random word appending, synonym replacement, and paraphrasing. We then create up to six versions of each original example by pairing combinations of answers with augmented versions of the questions.

Lastly, approximately 11% of the examples (N = 2,005,456) in our data derive from TAPEX (Liu et al., 2022), a dataset consisting of SQL generated from templates along with their corresponding execution result. To improve the quality and better align these data with downstream tasks we perform the following modifications. First, we remove 2.3 million duplicates (of the original 5 million examples), add a FROM clause to the SQL code with a fictitious table name using a random combination of 1-3 column names, and filter out any examples that could not be parsed by mo-sql-parsing[2]. Next, we train a SQL-to-Text model (T5-3B) on the Spider (Yu et al., 2019b) dataset in order to generate natural language statements for each SQL query.

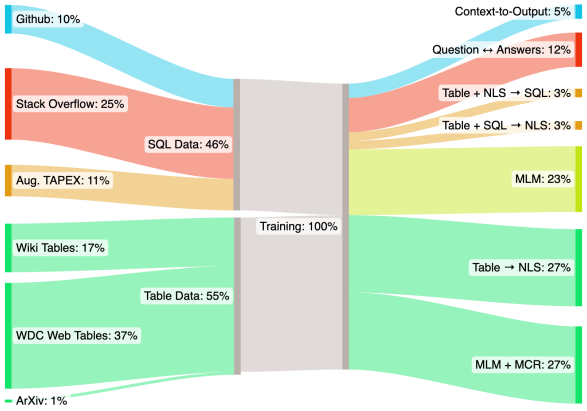

Figure 2: Pretraining data sources and objective mix ratios. *Left side:* Proportion of total pretraining data (N = 18,612,078) contributed by each data source. *Right side:* The corresponding objective mix during training when using an equal ratio of MLM-based to context-to-output objectives. Flow colors indicate which datasets feed into which objectives, except for MLM which is applied to all SQL data sources.

3.2 Objectives for Multi-Task Pretraining

MLM-Based Objectives A critical component in pretraining encoder-decoder models is a MLM-based objective. In STAMP we follow the span corruption style of MLM from Raffel et al. (2020), which involves replacing contiguous whole words

[1] As discussed in Appendix A, we consider accepted answers, favorite answers, or answers that received upvotes above some fixed threshold as latent human supervision.

[2] https://github.com/klahnakoski/mo-sql-parsing

from the text with sentinel tokens in the inputs, and then the decoder generates the replaced text preceded by the corresponding to sentinel token. We set the mean span to 3, with a denoising rate of 15% following the default T5 configuration. This span corruption objective is applied to sequences of NL and SQL code. For pretraining datasets that also include structured knowledge we apply the masked column recovery (MCR) objective, as introduced in Yin et al. (2020), which encourages the model to learn table schemas using the natural language statement and row information as context. In our implementation, 25% of the column names and data types (when available) are masked with a sentinel token. Note, only MCR is applied to the sequence containing the column names to avoid overlapping MLM and MCR masking. More concretely, let $\mathbf{x}^{\text{mask}} = (\mathbf{x}^{\text{MLM}}, \mathbf{x}^{\text{MCR}})$ be the input sequence combining MLM and MCR masking, then our masked span prediction loss $\mathcal{L}_M$ over a sequence of length T is:

$$\mathcal{L}_M(\theta) = \sum_{t=1}^{T} - \log P_\theta \left(x_t^{\text{mask}} \mid \mathbf{x}^{\setminus \text{mask}}, \mathbf{x}_{<t}^{\text{mask}} \right),$$

where x_t^{mask} is the masked token for the decoder to predict, $\mathbf{x}^{\setminus \text{mask}}$ is the encoded masked input, $\mathbf{x}_{<t}^{\text{mask}}$ is the sequence generated by the decoder up to token t, and θ are the model parameters.

Context-to-Output Objectives In addition to MLM-based objectives we include causal language modeling objectives (Radford et al., 2019; Liu et al., 2018), which partition sequences into contexts and outputs in order to mimic the format of many down-stream tasks. For unimodal datasets, such as GitHub SQL, we create the context and output by uniformly sampling a split point based on line-breaks within each code example. For tabular datasets we treat the table as input and the paired NL as output, thereby teaching the model to connect the structured and unstructured information.

For Stack Overflow, the natural partition between a question and each of the answers defines the context to output splits. We use the augmentations described in 3.1 to create additional unique question-to-answer pairs. We apply dual learning to better align the question prompt with the answer.

Finally, for trimodal data like our augmented-TAPEX we model Table + NL $\rightarrow$ SQL, or in the dual learning (Wang et al., 2021b) setting we model Table + SQL $\rightarrow$ NL. Thus for a sequence $\mathbf{x}$ of length T with a split point $S \in (0, T)$ that is either randomly selected or based a natural split in the data, we define the context-to-output loss $\mathcal{L}_{\text{C2O}}$ as:

$$\mathcal{L}_{\text{C2O}}(\theta) = \sum_{t=S}^{T} - \log P_\theta \left(y_t \mid \mathbf{z}, \mathbf{y}_{<t} \right),$$

where $\mathbf{z} = \mathbf{x}_{<S}$ is the left context and $\mathbf{y} = \mathbf{x}_{S\leq}$ the right output.

Combining Objectives Prior work shows the importance of MLM (Liu et al., 2019; Aroca-Ouellette and Rudzicz, 2020; Raffel et al., 2020) and the benefit of including a small percentage of context-to-output objectives. For instance, Tay et al. (2022) recommend approximately 20% of the objective mixture to be context-to-output. However, unlike Tay et al. (2022) we are not pretraining from scratch, rather we seek to build on existing checkpoints and hence we consider greater rates of context-to-output. In our implementation, we sample an objective per-example during pretraining, where the pool of objectives depends on the data source of each example. Hence, each training mini-batch combines examples from multiple data sources that are formatted as a mix of objectives. Figure 2 summarizes our dataset and objective mix, showing the connection between each input data source and a corresponding objective.

```
table : racetracks |
col : track | city | state |
      opened | surface | length
row 1 : altamont park | tracy | california |
        1966-2008 | asphalt | 0.5 miles
```

Figure 3: Row-centric format for tables.

```
<table> racetracks
<column_name> track<column_value> altamont park
<column_name> city<column_value> tracy
<column_name> state<column_value> california
<column_name> opened<column_value> 1966-2008
<column_name> surface<column_value> asphalt
<column_name> length<column_value> 0.5 miles
```

Figure 4: Column-centric format for tables.

3.3 Unified Format for Learning from Structured Knowledge

In order to bridge the gap between pretraining and downstream tasks, we explore unified formats for structured knowledge. Connecting NL to structured knowledge is challenging with limited data. A unified table format, however, allows the model to leverage learning from large scale pretraining

for smaller datasets. Moreover, in some cases Xie et al. (2022) report worse performance for multi-task versus single-task training, which we suspect is due to inconsistent formatting. Thus, we linearize structured knowledge into both row- and column-centric formats. Figure 3 shows the row-centric format, and Figure 4 shows the equivalent information in the column-centric format.

Lastly, we use special tokens in the encoder to preface each data modality (NL, structured knowledge, and SQL), and encourage sharing across tasks with common modalities. Additional tags prompt the decoder with the desired task, reflecting each of our objectives: MLM, table-to-text, SQL-to-SQL, Table and NL-to-SQL, Stack Overflow question answering, and dual learning variations.

4 Experiments

4.1 Evaluation Setup

We evaluate our pretrained checkpoints on SQL tasks following the UnifiedSKG framework (Xie et al., 2022). Specifically, for text-to-SQL benchmarking we evaluate on Spider without database row information (Yu et al., 2019b) and WikiSQL with row information (Zhong et al., 2017), as well as conversational text-to-SQL datasets SPaRC (Yu et al., 2019c) and CoSQL (Yu et al., 2019a), and in alignment with our bimodal objectives we also evaluate on SQL2Text (Shu et al., 2021). For each dataset we use pre-defined train, validation, and test splits. In Appendix C lists our evaluation settings, Appendix D contains details on the evaluation datasets, and Appendix E includes additional results.

4.2 Main Results

We present our main results in Table 1, with baseline results as reported in each comparison approach. We group models with SQL-specific decoders on top, and encoder-decoders like STAMP that have more general token decoders on bottom. Overall we find that our STAMP yields better results than domain specific solutions and text- or code-only pretrained models. SMBOP + GRAPPA (Rubin and Berant, 2021) is similar to our work with multi-task learning and additional pretraining, however they rely on a SQL specific parsing algorithm. Whereas, our framework focuses on larger, more diverse sources of structured knowledge and a complementary multi-task learning strategy.

We highlight that pretraining on structured information alone like TABERT (Yin et al., 2020), or a general code pretraining dataset like CodeT5 (Wang et al., 2021b) does not produce exceptional results on text-to-SQL. Likewise, a large multi-task learning approach like T0 performs worse than STAMP models and vanilla T5, indicating that the benefits of multi-task learning depend on having a degree of domain relevance. Specifically T0's multi-task learning approach, which centers on text-only domains, does not benefit SQL tasks. Lastly, despite constrained decoding being very different than our approach, we include results for PICARD (Scholak et al., 2021) because it is an extremely effective approach that complements STAMP.

4.3 Ablation Studies

Denoising versus Context-to-Output In Table 2 we report development set performance of STAMP models that build on the T5-base checkpoint. We train each model on our full row-centrically orientated dataset and only vary the objective mixture. Unlike prior work (Tay et al., 2022; Aroca-Ouellette and Rudzicz, 2020) that pretrains from scratch, during our additional structured knowledge pretraining we observe that higher rates context-to-output objectives tend to perform best.

At the extremes of the objective mix we see mixed results. Setting MLM / context-to-output ratios to 100% / 0%, improves performance on text-to-SQL — indicating the benefit from our pretraining data. However, on the other extreme, model performance suffers with no MLM and only context-to-output. Nonetheless, by combining the two objectives we see the best performance overall. Specifically, an equal mix of MLM and completion either throughout pretraining or after one epoch of entirely MLM training results in noticeably higher performance compared to vanilla T5.

Our results complement those in literature (Tay et al., 2022; Wang et al., 2021b; Aghajanyan et al., 2021; Aribandi et al., 2021; Sanh et al., 2022; FitzGerald et al., 2022), showing the importance of mixing additional objectives with MLM. Unlike Tay et al. (2022), however, our results show that higher rates of context-to-output are optimal, which we attribute to our approach of building on prior checkpoints and not pretraining from scratch.

Tables versus SQL Datasets Table 3 presents an ablation study comparing of STAMP and CodeSTAMP models trained on different pretrain-

	Model	# Params	Spider (EM ↑ / Exec ↑)	Sup. WikiSQL (EM ↑)	SParC (EM ↑)	CoSQL (EM ↑)	SQL2Text (BLEC ↑)
SQL-Grammar	Seq2SQL (Zhong et al., 2017)		—— / ——	49.5	–	–	–
	SQLNET (Xu et al., 2017)		—— / ——	63.2	–	–	–
	IRNet (Guo et al., 2019)		55.4 / ——	–	–	–	–
	RAT-SQL (Wang et al., 2021a)		62.7 / ——	–	–	–	–
	TABERT (Yin et al., 2020)	345M+	65.2 / ——	–	–	–	–
	SCoRE (Tao Yu et al., 2021)	500M	—— / ——	–	62.2	52.1	–
	BERT + RAT-SQL (Wang et al., 2021a)	500M	69.7 / ——	–	–	–	–
	RAT-SQL + GAP (Shi et al., 2020)		71.8 / ——	–	–	–	–
	SMBOP + GRAPPA (Rubin and Berant, 2021)		74.7 / 75.0	–	–	–	–
	T5 + PICARD† (Scholak et al., 2021)	3B	74.1 / 76.3	–	–	**56.9**	–
Encoder-Decoder	CodeT5 (Wang et al., 2021b)	770M	64.6 / ——	76.6	57.9	48.4	91.9
	T5 (from Xie et al. (2022))	770M	66.6 / 68.3	–	56.7	48.3	93.4
	T5 MT-P (Xie et al., 2022)	770M	67.6 / ——	–	59.0	51.6	**93.9**
	T0 (Sanh et al., 2022)	3B	68.1 / ——	–	–	–	92.9
	T5 (from Xie et al. (2022))	3B	71.8 / 74.4	–	61.5	54.1	92.7
	(ours) STAMP-Large RC	770M	71.6 / 74.4	78.9	61.4	53.7	93.0
	(ours) STAMP-Large CC	770M	71.8 / 76.3	79.3	59.6	51.4	93.3
	(ours) CodeSTAMP-Large RC	770M	70.7 / 74.5	84.3	58.8	50.6	92.0
	(ours) CodeSTAMP-Large CC	770M	69.4 / 72.8	**84.7**	58.7	52.0	92.1
	(ours) STAMP-3B RC	3B	**75.2 / 78.0**	79.4	**64.4**	56.4	92.6

Table 1: Development set performance on text-to-SQL benchmarks for both T5, CodeT5, and our results with additional pretraining on our structured knowledge. All STAMP checkpoints train with a 50/50 mixture of context-to-output and MLM-based objectives. STAMP results are separated by variations in the pretraining data, specifically *CC* and *RC* denote column- and row-centric table formats, respectively, and *w/ Tables* denotes the full pretraining dataset whereas *SQL-only* is a subset that omits the NL+Table datasets. Note: A dagger (†) indicates constrained decoding approach, which is complementary but not used in our work, models in *italics* are our work.

%-MLM in Objective Mix	Spider (Exec ↑)	Sup. WikiSQL (EM ↑)	SParC (EM ↑)	CoSQL (EM ↑)	SQL2Text (BLEC ↑)
100%	63.2	78.2	52.0	43.8	93.6
75%	64.0	78.1	52.3	44.0	93.2
50%	**64.5**	77.9	51.9	**44.5**	93.2
100→50%	62.9	**78.9**	**52.4**	42.3	**94.2**
0%	61.3	78.0	49.6	40.1	93.0
Vanilla T5-base	60.1	74.1	49.9	42.4	93.7

Table 2: Development set performance for T5-base, and base-sized STAMP models pretrained on our full row-centric dataset with varying objective mixes. For each pretrained STAMP model we specify the proportion of training examples using the MLM-based objective, with the remaining examples using a dataset-specific context-to-output objective. We also explore dynamic mixing ratios, where 100→50% represents training with 100% MLM in the first epoch, followed by a 50%/50% mix of during the remaining epochs.

ing data. Specifically, we report the effect of pretraining on data where tables are in the row-centric (RC) versus column-centric (CC) format. We also explore the effect of pretraining on all data versus only SQL-related data. Overall we see that pretraining on all datasets generally improves performance on text-to-SQL — confirming the finding of Yin et al. (2020) that aligning NL and tables improves performance. Moreover, in comparison with Yin et al. (2020), our results show that adding SQL code to the data mix further boosts performance.

Row-Centric versus Column-Centric We preprocess the pretraining and benchmark datasets from UnifiedSKG (Xie et al., 2022) with consistent table formatting. Row-centric formats are more similar to natural language and do not require learning any new special tokens, which better leverages the original NL pretraining of T5. Whereas, the column-centric format requires special tokens that preface the table, columns, and each value in a column. While new special tokens must be learned from scratch, we hypothesized that the column-centric format is advantageous since text-to-SQL is inherently more column and schema oriented and often not dependent on row information. Surprisingly, Table 3 shows no clear advantage for either RC or CC formats. In fact, the mixed results hold for even across model sizes (Large vs Base) and initial pretraining (T5 vs CodeT5). Our results suggest that further pretraining on enough high-quality data helps to nullify the advantages or disadvantages of each table linearization method.

T5 versus CodeT5 as Starting Point Table 3 shows the high performance of base-sized CodeT5

Starting Checkpoint	Additional STAMP Pretraining Data	Spider (Exec ↑)	Sup. WikiSQL (EM ↑)	SParC (EM ↑)	CoSQL (EM ↑)	SQL2Text (BLEC ↑)
T5-Large	—	71.7	75.3	57.4	48.8	93.4
T5-Large	RC, w/ Tables	74.4	78.9	**61.4**	**53.7**	93.0
T5-Large	RC, SQL-only	72.8	79.5	60.1	51.4	**93.6**
T5-Large	CC, w/ Tables	**76.3**	79.3	59.6	51.4	93.3
T5-Large	CC, SQL-only	74.5	79.1	51.9	50.9	93.3
CodeT5-Large	—	68.4	76.6	57.9	48.4	91.9
CodeT5-Large	RC, w/ Tables	71.9	84.4	59.7	50.9	92.1
CodeT5-Large	CC, w/ Tables	72.8	**84.7**	58.7	52.0	92.1
T5-Base	—	60.8	74.1	49.9	42.4	93.7
T5-Base	RC, w/ Tables	64.5	77.9	51.9	44.5	93.2
T5-Base	RC, SQL-only	61.7	77.8	52.4	42.8	93.4
T5-Base	CC, w/ Tables	60.5	79.5	49.9	41.3	93.9
T5-Base	CC, SQL-only	59.2	79.5	46.8	38.9	**94.0**
CodeT5-Base	—	67.1	76.0	54.4	47.2	93.5
CodeT5-Base	RC, w/ Tables	69.0	83.5	**55.6**	**47.7**	92.9
CodeT5-Base	CC, w/ Tables	**69.2**	**84.5**	54.7	46.9	93.4

Table 3: Development set performance on SQL benchmarks for both the original T5-base, T5-large, CodeT5-base, and CodeT5-large checkpoints, as well as our results with additional pretraining on our structured knowledge pretraining dataset. All STAMP checkpoints train with a 50/50 mixture of context-to-output and MLM-based objectives. STAMP results are separated by variations in the pretraining data, specifically *CC* and *RC* denote column- and row-centric table formats, respectively, and *w/ Tables* denotes the full pretraining dataset described in 3 whereas the *SQL-only* subset omits the Text+Table datasets. The best performer at each model size is shown in **bold**.

and CodeSTAMP models. Relative to their T5[3] and STAMP counterparts, the base-sized CodeT5 and CodeSTAMP models show significant performance gains across all text-to-SQL benchmarks. In particular, models based on the CodeT5-base checkpoint show exceptional performance when given row information in the tables, as is the case for WikiSQL. Interestingly, models based on CodeT5 do not exhibit the same performance gains compared to those based on T5 for large-sized models. In fact, models based on CodeT5-large only excel at WikiSQL, whereas models based on T5-large excel in all other tasks. We hypothesize that large-sized models based on CodeT5 do not outperform their peers in the same way as the base-sized models due to scaling issues caused by CodeT5's much smaller CodeSearchNet (Husain et al., 2020) pretraining dataset, especially when using a smaller dataset to train the larger model. Additionally, we see that models based on CodeT5 checkpoints tend to perform worse on SQL2Text, which is likely because natural language in CodeT5's original pretraining data is limited to comments in code, and hence the ability to generate natural language may be underdeveloped relative to T5.

[3]Our results for T5-Large on Spider, SParC, and CoSQL differ from Xie et al. (2022) and Scholak et al. (2021). On Spider we achieve 3.4%-points higher than Xie et al. (2022), and 4.5%-points higher than Scholak et al. (2021). In our implementation we use a maximum input sequence length of 1024 and an output sequence lengths of 256 to avoid truncation.

5 Conclusion

We present STAMP, a pretraining framework for encoder-decoders on SQL tasks. We introduce a large scale pretraining dataset of tables, SQL code, discussions on Stack Overflow, and a modified TAPEX dataset (Liu et al., 2022). We complement our data with a multi-task learning framework to align the data modalities, finding that an equal mix of the objectives is optimal. We explore both row- and column-centric approaches to linearizing tables, creating a unified format across training stages. A column-centric format is often superior, challenging the conventional row-centric approach. Lastly, while PL pretraining may help generalization (Athiwaratkun et al., 2022), STAMP models based on T5 yield better performance.

6 Limitations

While our work displays many strengths, we highlight some important limitations in our analysis. Namely, we pretrain our STAMP models on a range of sources containing structured knowledge, however our analysis is limited to text-to-SQL tasks and does not demonstrate if such pretraining helps more generally in structured information tasks. For instance, STAMP pretrains on tables with (1) masked column recovery as a way to learn the structure of a table using the rows and natural language statement as context, and (2) a context-to-output objective that always includes the table in the context (when available) — since this matches the format of text-to-SQL tasks. It is unclear if our objective choices for pretraining on tables perform equally well on the range of structured knowledge tasks, such as table question-answering, table summarization, data-to-text, fact verification, and others explored in Xie et al. (2022). Second, we acknowledge that significant GPU resources are required for pretraining, even in continued pretraining approaches like ours which limit the breadth of ablations studies. Conversely, our work explores pretraining at smaller scales where certain phenomena like strong zero-shot performance is unlikely. Pretraining specifically on structured knowledge has an unknown value at larger scales with models having tens or hundreds of billions of parameters.

7 Ethics Statement

We acknowledge the importance of the ACL Ethics Policy and agree with it. Large language models can appear confident while providing false information. In our work we are fortunate that incorrect SQL output is verifiable and take care to report the true reliability of the systems. Additionally we acknowledge that large language models, such as those studied in this work, may generate toxic language (Gehman et al., 2020). While we avoid pretraining on data sources and content from web domains with offensive language, we acknowledge that even our data gathered from reputable publishers introduces bias (Bolukbasi et al., 2016).

Acknowledgements

We would like to thank Henry Zhu for providing a sql-to-text model that we used to augment TAPEX with natural language statements.

References

Armen Aghajanyan, Anchit Gupta, Akshat Shrivastava, Xilun Chen, Luke Zettlemoyer, and Sonal Gupta. 2021. Muppet: Massive Multi-task Representations with Pre-Finetuning.

Rami Aly, Zhijiang Guo, Michael Sejr Schlichtkrull, James Thorne, Andreas Vlachos, Christos Christodoulopoulos, Oana Cocarascu, and Arpit Mittal. 2021. The Fact Extraction and VERification Over Unstructured and Structured information (FEVEROUS) Shared Task. In *Proceedings of the Fourth Workshop on Fact Extraction and VERification (FEVER)*, pages 1–13, Dominican Republic. Association for Computational Linguistics.

Ewa Andrejczuk, Julian Martin Eisenschlos, Francesco Piccinno, Syrine Krichene, and Yasemin Altun. 2022. Table-to-text generation and pre-training with tabt5.

Vamsi Aribandi, Yi Tay, Tal Schuster, Jinfeng Rao, Huaixiu Steven Zheng, Sanket Vaibhav Mehta, Honglei Zhuang, Vinh Q. Tran, Dara Bahri, Jianmo Ni, Jai Gupta, Kai Hui, Sebastian Ruder, and Donald Metzler. 2021. ExT5: Towards Extreme Multi-Task Scaling for Transfer Learning. *arXiv:2111.10952 [cs]*.

Stephane Aroca-Ouellette and Frank Rudzicz. 2020. On Losses for Modern Language Models. *arXiv:2010.01694 [cs]*.

Ben Athiwaratkun, Sanjay Krishna Gouda, Zijian Wang, Xiaopeng Li, Yuchen Tian, Ming Tan, Wasi Uddin Ahmad, Shiqi Wang, Qing Sun, Mingyue Shang, Sujan Kumar Gonugondla, Hantian Ding, Varun Kumar, Nathan Fulton, Arash Farahani, Siddhartha Jain, Robert Giaquinto, Haifeng Qian, Murali Krishna Ramanathan, Ramesh Nallapati, Baishakhi Ray, Parminder Bhatia, Sudipta Sengupta, Dan Roth, and Bing Xiang. 2022. Multi-lingual Evaluation of Code Generation Models.

Tolga Bolukbasi, Kai-wei Chang, James Zou, Venkatesh Saligrama, and Adam Kalai. 2016. Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings. *arXiv preprint arXiv:1607.06520v1*, (Nips):1–25.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners.

Jifan Chen, Yuhao Zhang, Lan Liu, Rui Dong, Xinchi Chen, Patrick Ng, William Yang Wang, and Zhiheng Huang. 2022. Improving Cross-task Generalization

of Unified Table-to-text Models with Compositional Task Configurations.

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.

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Wang. 2021. HybridQA: A Dataset of Multi-Hop Question Answering over Tabular and Textual Data.

Zhoujun Cheng, Haoyu Dong, Ran Jia, Pengfei Wu, Shi Han, Fan Cheng, and Dongmei Zhang. 2022. FORTAP: Using Formulas for Numerical-Reasoning-Aware Table Pretraining.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. PaLM: Scaling Language Modeling with Pathways. *arXiv:2204.02311 [cs]*.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*.

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, Pretraining Objectives, and Downstream Tasks. *arXiv:2201.09745 [cs]*.

Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten Bosma, Zongwei Zhou, Tao Wang, Yu Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc V. Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. 2022. GLaM: Efficient Scaling of Language Models with Mixture-of-Experts.

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

Jack FitzGerald, Shankar Ananthakrishnan, Konstantine Arkoudas, Davide Bernardi, Abhishek Bhagia, Claudio Delli Bovi, Jin Cao, Rakesh Chada, Amit Chauhan, Luoxin Chen, Anurag Dwarakanath, Satyam Dwivedi, Turan Gojayev, Karthik Gopalakrishnan, Thomas Gueudre, Dilek Hakkani-Tur, Wael Hamza, Jonathan J. Hüser, Kevin Martin Jose, Haidar Khan, Beiye Liu, Jianhua Lu, Alessandro Manzotti, Pradeep Natarajan, Karolina Owczarzak, Gokmen Oz, Enrico Palumbo, Charith Peris, Chandana Satya Prakash, Stephen Rawls, Andy Rosenbaum, Anjali Shenoy, Saleh Soltan, Mukund Harakere Sridhar, Lizhen Tan, Fabian Triefenbach, Pan Wei, Haiyang Yu, Shuai Zheng, Gokhan Tur, and Prem Natarajan. 2022. Alexa Teacher Model: Pretraining and Distilling Multi-Billion-Parameter Encoders for Natural Language Understanding Systems. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2893–2902, Washington DC USA. ACM.

Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A. Smith. 2020. RealToxicityPrompts: Evaluating Neural Toxic Degeneration in Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3356–3369, Online. Association for Computational Linguistics.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation.

Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. TAPAS: Weakly Supervised Table Parsing via Pre-training. *arXiv:2004.02349 [cs]*.

Sepp Hochreiter and Jurgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2020. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. 2020. SpanBERT: Improving Pre-training by Representing and Predicting Spans.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *ICLR*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. *arXiv:1910.13461 [cs, stat]*.

Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. 2018. Generating Wikipedia by Summarizing Long Sequences.

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.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv:1907.11692 [cs]*.

Linyong Nan, Chiachun Hsieh, Ziming Mao, Xi Victoria Lin, Neha Verma, Rui Zhang, Wojciech Kryściński, Hailey Schoelkopf, Riley Kong, Xiangru Tang, Mutethia Mutuma, Ben Rosand, Isabel Trindade, Renusree Bandaru, Jacob Cunningham, Caiming Xiong, Dragomir Radev, and Dragomir Radev. 2022. FeTaQA: Free-form Table Question Answering. *Transactions of the Association for Computational Linguistics*, 10:35–49.

Linyong Nan, Dragomir Radev, Rui Zhang, Amrit Rau, Abhinand Sivaprasad, Chiachun Hsieh, Xiangru Tang, Aadit Vyas, Neha Verma, Pranav Krishna, Yangxiaokang Liu, Nadia Irwanto, Jessica Pan, Faiaz Rahman, Ahmad Zaidi, Mutethia Mutuma, Yasin Tarabar, Ankit Gupta, Tao Yu, Yi Chern Tan, Xi Victoria Lin, Caiming Xiong, Richard Socher, and Nazneen Fatema Rajani. 2021. DART: Open-Domain Structured Data Record to Text Generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 432–447, Online. Association for Computational Linguistics.

Ankur P. Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. 2020. ToTTo: A Controlled Table-To-Text Generation Dataset.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. *JMLR*, page 9.

Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, Fei Huang, and Yongbin Li. 2022. A Survey on Text-to-SQL Parsing: Concepts, Methods, and Future Directions.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models Are Unsupervised Multitask Learners. page 24.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *arXiv:1910.10683 [cs, stat]*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *EMNLP*.

Ohad Rubin and Jonathan Berant. 2021. SmBoP: Semi-autoregressive Bottom-up Semantic Parsing. In *NAACL-HLT*. arXiv.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, Manan Dey, M. Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Tali Bers, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M. Rush. 2022. Multitask Prompted Training Enables Zero-Shot Task Generalization. *arXiv:2110.08207 [cs]*.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. *arXiv:2109.05093 [cs]*.

Peng Shi, Patrick Ng, Zhiguo Wang, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Cicero Nogueira dos Santos, and Bing Xiang. 2020. Learning Contextual Representations for Semantic Parsing with Generation-Augmented Pre-Training. *arXiv:2012.10309 [cs]*.

Chang Shu, Yusen Zhang, Xiangyu Dong, Peng Shi, Tao Yu, and Rui Zhang. 2021. Logic-Consistency Text Generation from Semantic Parses. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4414–4426, Online. Association for Computational Linguistics.

Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V. Le. 2017. Don't Decay the Learning Rate, Increase the Batch Size.

Tao Yu, Rui Zhang, Alex Polozov, Christopher Meek, and Ahmed Hassan Awadallah. 2021. SCoRe: Pre-Training for Context Representation in Conversational Semantic Parsing. In *ICLR*.

Yi Tay, Mostafa Dehghani, Vinh Q. Tran, Xavier Garcia, Jason Wei, Xuezhi Wang, Hyung Won Chung, Dara Bahri, Tal Schuster, Huaixiu Steven Zheng, Denny Zhou, Neil Houlsby, and Donald Metzler. 2022. UL2: Unifying Language Learning Paradigms.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, YaGuang Li, Hongrae Lee, Huaixiu Steven Zheng, Amin Ghafouri, Marcelo Menegali, Yanping Huang, Maxim Krikun, Dmitry Lepikhin, James Qin, Dehao Chen, Yuanzhong Xu, Zhifeng Chen, Adam Roberts, Maarten Bosma, Vincent Zhao, Yanqi Zhou, Chung-Ching Chang, Igor Krivokon, Will Rusch, Marc Pickett, Pranesh Srinivasan, Laichee Man, Kathleen Meier-Hellstern, Meredith Ringel Morris, Tulsee Doshi, Renelito Delos Santos, Toju Duke, Johnny Soraker, Ben Zevenbergen, Vinodkumar Prabhakaran, Mark Diaz, Ben Hutchinson, Kristen Olson, Alejandra Molina, Erin Hoffman-John, Josh Lee, Lora Aroyo, Ravi Rajakumar, Alena Butryna, Matthew Lamm, Viktoriya Kuzmina, Joe Fenton, Aaron Cohen, Rachel Bernstein, Ray Kurzweil, Blaise Aguera-Arcas, Claire Cui, Marian Croak, Ed Chi, and Quoc Le. 2022. LaMDA: Language Models for Dialog Applications.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2021a. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. *arXiv:1911.04942 [cs]*.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven C. H. Hoi. 2021b. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation.

Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. 2021c. TUTA: Tree-based Transformers for Generally Structured Table Pre-training. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1780–1790.

Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I. Wang, Victor Zhong, Bailin Wang, Chengzu Li, Connor Boyle, Ansong Ni, Ziyu Yao, Dragomir Radev, Caiming Xiong, Lingpeng Kong, Rui Zhang, Noah A. Smith, Luke Zettlemoyer, and Tao Yu. 2022. UnifiedSKG: Unifying and Multi-Tasking Structured Knowledge Grounding with Text-to-Text Language Models. *arXiv:2201.05966 [cs]*.

Xiaojun Xu, Chang Liu, and Dawn Song. 2017. SQLNet: Generating Structured Queries From Natural Language Without Reinforcement Learning.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Richard Socher, and Caiming Xiong. 2021. GraPPa: Grammar-Augmented Pre-Training for Table Semantic Parsing.

Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong, Richard Socher, Walter Lasecki, and Dragomir Radev. 2019a. CoSQL: A Conversational Text-to-SQL Challenge Towards Cross-Domain Natural Language Interfaces to Databases. 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 1962–1979, Hong Kong, China. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2019b. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task.

Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, Emily Ji, Shreya Dixit, David Proctor, Sungrok Shim, Jonathan Kraft, Vincent Zhang, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019c. SParC: Cross-Domain Semantic Parsing in Context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4511–4523, Florence, Italy. Association for Computational Linguistics.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning.

A Pretraining Dataset Details

A.1 Stack Overflow Augmentations

We perform several augmentation steps on Stack Overflow examples to construct our pretraining dataset. Our first step is to create four augmented versions of each question using random word deletion, random word appending, synonym replacement, and paraphrasing. Next, we create up to five different combinations of input-label pairs by re-arranging the answers.

For some pertinent background on Stack Overflow, each example consists of a question and one or more answers. The user who answered the question can mark the answer that solved their problem as correct, and other users can upvote answers that they found useful as well.

Let N be the number of answers for a question. The following strategies are used to create the labels for the augmented examples:

1. The accepted answer (if there is one)
2. The most upvoted answer if it has been upvoted more than the accepted answer

3. Concatenation of all answers

4. Randomly select an answer A_i and append all answers up to and including that one to the question, then use the concatenation of all $A_{i+1}, A_{i+2} \ldots A_N$ answers as the label

5. Randomly select an answer, A_i, and append all answers up to and including that one to the question. Randomly select another answer, A_k, from the remaining $A_{i+1}, A_{i+2} \ldots A_N$ answers and use the concatenation of all $A_k, A_{k+1} \ldots A_N$ answers as the label

Each of these strategies is constrained by a total sequence length of 1024 tokens. If we need to truncate any tokens, we truncate them in the following order:

1. Text in Answer

2. Code in Question

3. Text in Question

Our intuition is that this is the order of least important to most important to preserve the logical relationship between question and answer, with code in the answer being the most critical (which is never truncated).

A.2 Data Filtering

As briefly mentioned in 3.1, we filter noisy examples from both the table and SQL dataset. Below we provide more details on this pre-processing step.

Table Filtering Since table data is often web-scraped it contains many noisy examples. Specifically, examples where the table information has a tenuous relation to the paired natural language statement. Moreover, since our initial collection of raw data was much larger for table sources versus SQL source, we chose to implement a filtering approach to reduce these noisy examples. Specifically, we first calculate the edit-similarity between each sample's table and the NL statement, after removing special tokens or tags. We then compute the same metric on ToTTo, which is a high-quality table-to-text benchmark, and qualitatively chose our filtering threshold as 50.0 which is slightly lower than ToTTo's average edit-similarity. All samples from our Wiki, Web, and ArXiv tables datasets with an edit-similarity below 50.0 are removed. In total we remove approximately 74% of samples from the raw data.

Github SQL Filtering For the Github SQL data we again see a large proportion of noisy or repetitive samples in the raw data. Specifically, Github SQL data can contain many repetitive statements within one sample, such as thousands of consecutive INSERT statements that data into a table. The insert statements are often either very repetitive, or contain very noisy information like compressed images, PDFs, or spatial objects. Our filtering method largely consists of using regular expression to identify such repetitive statements. After finding long sequences of insert statements we keep only a random sample of 10 insert statements if the insert statements are repetitive but not overly long or unreadable. However, we remove all insert statements that load noisy information into a table. In total the number of samples staying approximately the same, however we reduce the size of the dataset by approximately 61%.

A.3 Pretraining Dataset Statistics

In Table 4 we provide summary statistics for the pretraining dataset, including each of the SQL and Table subsets. Raw document counts help to show the amount of filtering applied to the raw data in order reduce noisy and potentially detrimental samples, whereas the final training sample counts show the training dataset size after tokenizing and partitioning documents into sequences.

B Pretraining Hyperparameters

Batch size. For 3B and large models we train for at a small batch size of 64 for the first epoch, then for most of the second and third epoch we double the batch size to 128, and then for the final 5-10% of training we double the batch size again to 256. Starting with a small batch size provides better gradient efficiency, while larger batch sizes give more precise gradient estimates which is beneficial later in training (Smith et al., 2017). For base sized models we opt for a batch size of 128 for all three epochs before the cooldown period.

Sequence length. Data are pre-processed and tokenized offline into sequences of at most 1024 tokens. We do not pack inputs, and instead use one example in per input and then pad accordingly. For the larger T5-3B model we found that training for the first 75-90% of steps on data pre-processed into a shorter max sequence length of 768 or 896, and then the remainder of training on data with 1024 tokens provided improved computational efficiency

Data Source	Modalities	Num. Raw Documents Initial (K)	Num. Raw Documents Filtered (K)	Num. Training Samples (K)	Avg. Number of Tokens Context	Avg. Number of Tokens Output	Avg. Number of Tokens Total
Github SQL	SQL	1,026	1,019	1,918	280 ±272	283 ±273	563 ±545
Stack Overflow	NL, SQL	1,670	1,631	4,480	318 ±215	289 ±192	607 ±407
Aug. TAPEX	NL, Table, SQL	2,165	2,165	2,005	471 ±210	30 ±14	501 ±224
Wiki Tables	NL, Table	6,350	3,080	3,080	148 ±117	98 ±68	246 ±185
Web Tables	NL, Table	32,295	7,032	7,032	142 ±79	132 ±97	274 ±176
ArXiv Tables	NL, Table	119	24	24	275 ±154	184 ±141	459 ±295
Full Dataset	NL, Table, SQL	43,766	14,991	18,612	189 ±161	149 ±139	338 ±300

Table 4: STAMP Pretraining dataset statistics by source. After the raw documents are filtered, we create training examples by partitioning documents into sequences of 1024 tokens which can result in more training samples than the initial set of raw documents. In the case of Stack Overflow we also augment the data creating a much larger collection of training samples from the initial pool of documents. Note: Raw document counts and final number of training samples are listed in thousands (K), the final pretraining dataset contains 18,612,078 samples.

without a discernible degradation in performance. Encoder inputs begin with a special token indicating the data modality, and the decoder inputs begin with a special token indicating the desired task. All sequences end with the same end of sequence token as Raffel et al. (2020).

Optimization. All models are pretrained with the AdamW (Kingma and Ba, 2015) optimizer, using an initial learning rate of 1e-4, and set momentum of $\beta_1 = 0.9$ and $\beta_2 = 0.98$. Our learning rate warms-up linearly over the first 1% of training steps, and then decays following a fixed cosine annealing schedule to 1e-7 after approximately 3 epochs. We set a gradient norm clipping with a maximum gradient norm of 1.0 (Pascanu et al., 2013). We train models based on T5 (Raffel et al., 2020) using the bf16 data type, whereas for models based on CodeT5 (Wang et al., 2021b) we use the fp16 data type in order to match the data type from prior pretraining. We fix the weight decay to 0.01 for all models.

C Evaluation Settings

For finetuning we follow the experimental setup of UnifiedSKG (Xie et al., 2022). Specifically, we use the Adafactor optimizer with decaying learning rate that is initially set to 5e-5, we set the batch size to 32, train for up to 200 epochs, and generate sequences using a beam size of 1. However, for WikiSQL we set a batch of 128, train for a maximum of 100 epochs, and use a beam size of 4. We use the same maximum lengths for the input and output as UnifiedSKG, except for Spider, SParC, and CoSQL where we increase input maximum length to 1024 and output to 256 sentence piece tokens to avoid truncating the inputs or outputs.

Pretrained Model	Finetune Method	Spider (Exec ↑)	Sup. WikiSQL (EM ↑)	SParC (EM ↑)	CoSQL (EM ↑)
STAMP-RC	STF	74.4	78.9	61.4	53.7
STAMP-RC	MTF	74.0	78.6	**61.9**	**55.0**
STAMP-CC	STF	76.3	79.3	59.6	51.4
STAMP-CC	MTF	73.9	79.1	**61.3**	**54.2**
CodeSTAMP-RC	STF	74.5	84.3	58.8	50.6
CodeSTAMP-RC	MTF	73.3	83.9	**59.4**	**51.9**
CodeSTAMP-CC	STF	72.8	84.7	58.7	52.0
CodeSTAMP-CC	MTF	71.3	83.5	58.3	50.8

Table 5: Development set performance on text-to-SQL benchmarks for large sized T5, STAMP CodeT5, and CodeSTAMP that are either Single-Task Finetuned (STF) on each dataset individually, or Multi-Task Finetuned (MTF) on all text-to-SQL datasets simultaneously. All STAMP checkpoints are pretrained with a 50/50 mixture of context-to-output and MLM-based objectives on the full pretraining dataset. STAMP results differentiated by whether they're trained with column- *CC* or row-centric *RC* table formats. We highlight results where multi-task finetuning outperforms single-task finetuning on an equivalent model in **bold**.

D Evaluation Datasets

We evaluate our model on each of the aforementioned datasets using the standard metrics for each task. We use the standard train, validation, and test splits for each of the datasets.

Spider The Spider dataset has 10,181 question-query pairs with queries using 200 databases representing 138 different domains and tables that are joined via foreign keys. We use the standard training and development splits, where training, development, and test sets have a 7:1:2 ratio, and each database appears in only one set (Yu et al., 2019b).

Fully Supervised WikiSQL The WikiSQL dataset has 80,564 question-query pairs, involving over 30,000 tables from Wikipedia (Zhong et al.,

Model	# Params	Spider† (EM ↑ / Exec ↑)	Sup. WikiSQL (EM ↑: Dev / Test)	SParC† (EM ↑ / Exec ↑)	CoSQL† (EM ↑ / Exec ↑)	SQL2Text (BLEC ↑: Dev / Test)
(ours) STAMP-Large RC	770M	71.6±0.3 / 75.0±0.9	78.8±0.2 / 79.5±0.2	60.9±0.5 / 66.0±0.6	53.7±0.3 / 61.9±0.5	93.5±0.4 / 94.8±0.3
(ours) STAMP-Large CC	770M	71.4±0.4 / 74.9±1.4	79.0±0.5 / 79.7±0.2	59.8±0.2 / 64.3±0.2	51.8±0.5 / 59.5±0.7	93.4±0.6 / 93.7±0.5
(ours) CodeSTAMP-Large RC	770M	70.5±0.3 / 74.3±0.3	84.3±0.1 / 84.3±0.3	59.1±0.4 / 63.5±0.8	51.5±1.1 / 59.7±0.6	92.2±0.4 / 91.8±0.3
(ours) CodeSTAMP-Large CC	770M	68.3±1.2 / 72.0±1.5	84.5±0.2 / 84.6±0.1	58.0±0.6 / 62.8±1.0	51.6±0.5 / 58.5±0.1	92.3±0.2 / 94.3±1.7
(ours) STAMP-3B RC	3B	74.3±1.1 / 78.0±0.3	79.4±0.1 / 80.0±0.1	63.9±1.0 / 68.7±1.2	56.2±1.0 / 66.1±1.2	92.8±0.4 / 93.2±0.6

Table 6: Average performance on SQL benchmarks over three finetuning runs with standard deviations. All STAMP checkpoints train with a 50/50 mixture of context-to-output and MLM-based objectives. STAMP results are separated by variations in the pretraining data, specifically *CC* and *RC* denote column- and row-centric table formats, respectively, and *w/ Tables* denotes the full pretraining dataset whereas *SQL-only* is a subset that omits the NL+Table datasets. Note: A dagger (†) indicates datasets where only a development set is available for assessing variance in performance, and models in *italics* are our work.

2017). We use the standard train, validation, and test splits for WikiSQL, providing 56,355 examples are set reserved for training. Note each table in WikiSQL is present in exactly one of the data splits.

SParC The SParC dataset consists of 4,298 question sequences with 12,726 question-SQL pairs and 200 databases spanning 138 domains (Yu et al., 2019c). SParC is built on Spider, however for SParC the question sequence is based on asking inter-related Spider questions. The question sequences are then paired with a manually annotated SQL query. Similar to Spider a 7:1:2 ratio is used to split the data into training, development, and test sets.

CoSQL The CoSQL dataset consists of 30k+ turns and 10k+ corresponding SQL queries along with 200 complex databases belonging to 138 domains, representing a large-scale cross-domain conversational setting (Yu et al., 2019a). Conversations are presented as between a user and a system, where the user provides a natural language description of a data table and the system must generate the corresponding SQL query. The conversational style of the dataset simulates the process of users asking clarifying questions to the system. Similar to Spider and SParC, CoSQL splits data into training, development and test sets with a ratio of 7:1:2, where each database appears in only one data split.

SQL2Text The SQL2Text dataset consists of 5,600, 1,400, and 1,034 train, development, and test examples, respectively (Shu et al., 2021). The dataset is natural language descriptions paired with their corresponding SQL queries.

E Additional Results

Single- versus Multi-Task Learning We explore the benefits of finetuning and evaluating either individually on each dataset (Single-Task Finetuning, STF) versus finetuning on all of the text-to-SQL benchmarks simultaneously then evaluating (Multi-Task Finetuning, MTF). For multi-task finetuning we balance the size of different datasets during training using the temperature up-sampling method proposed in Xie et al. (2022) and set the temperature to 2. The results of the ablation are presented in Table 5. We find mixed the results of multi-task finetuning. In almost every model MTF results in noticably better performance on the conversational SQL datasets SParC and CoSQL, however results for Spider and WikiSQL are slightly worse. We suspect that the close similarity between SParC and CoSQL explains the mutual benefit of multi-task finetuning. On the other hand, Spider uses a schema-only input format, whereas WikiSQL includes database content and is typically less difficult than Spider.

Performance Confidence Intervals In Table 6 we report more a more detailed look at our main results. Specifically, we report the average performance of our models over three finetuning runs and list the standard deviation in the performances.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
6

☑ A2. Did you discuss any potential risks of your work?
7

☑ A3. Do the abstract and introduction summarize the paper's main claims?
1

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☑ Did you use or create scientific artifacts?

Left blank.

☑ B1. Did you cite the creators of artifacts you used?
2

☒ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
We are first awaiting legal approval for distribution.

☑ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
3

☒ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
Data will not be released and models are only for research purposes.

☐ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
Not applicable. Left blank.

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
Available in Appendix.

C ☑ Did you run computational experiments?

4

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
4, Appendix

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
4, Appendix

☒ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
Unable to complete due to constraints on compute budget.

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
3

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
No response.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
No response.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
No response.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
No response.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
No response.