ToolWriter: Question Specific Tool Synthesis for Tabular Data

Anonymous NAACL-HLT 2021 submission

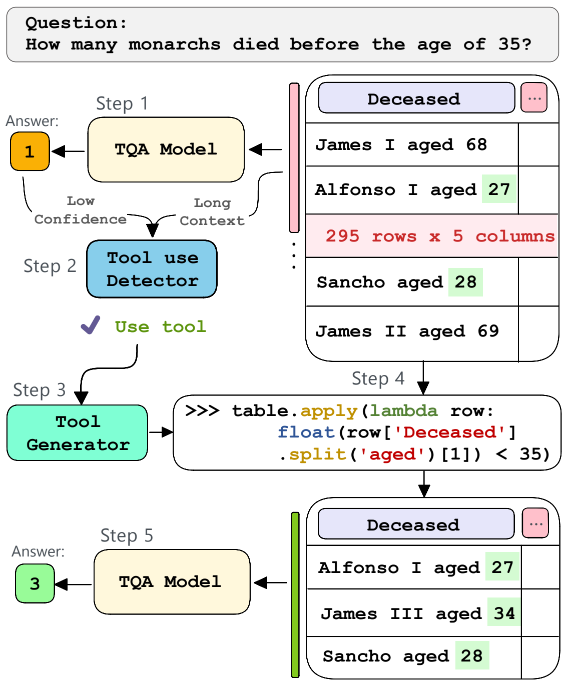
Abstract

Tabular question answering (TQA) presents a challenging setting for neural systems by requiring joint reasoning of natural language with large amounts of semi-structured data. Unlike humans who use programmatic tools like filters to transform data before processing, language models in TQA process tables directly, resulting in information loss as table size increases. In this paper we propose ToolWriter to generate query specific programs and detect when to apply them to transform tables and align them with the TQA model's capabilities. Focusing ToolWriter to generate row-filtering tools improves the state-of-the-art for WikiTableQuestions and WikiSQL with the most performance gained on long tables. By investigating headroom, our work highlights the broader potential for programmatic tools combined with neural components to manipulate large amounts of structured data.

1 Introduction

An important area for research in large language models (T5, PaLM, GPT-3) is combining them with "tools" to enhance their capabilities in question answering(Schick et al., 2023; Gao et al., 2022a; Parisi et al., 2022; Lazaridou et al., 2022). Tool-augmented approaches enable language models to externalize knowledge and computation by making explicit calls to APIs. However, these approaches do not process semi-structured data. We show that current models degrade in effectiveness significantly when questions and data become long and complex. A key task that demonstrates these limitations is tabular question answering (TQA) where long tables and complex questions are particularly challenging for current models.

Tabular question answering is a task in natural language processing that involves leveraging information from a semi-structured table to answer multi-hop compositional questions. It discourages purely symbolic approaches due to latent structure



```
>>> table.apply(lambda row:
        float(row['Deceased']
        .split('aged')[1]) < 35)
```

Figure 1: ToolWriter for tabular question answering introduces: 1) A tool-use detector; 2) A tool generator. Here a row-filter tool is generated as a program that transforms semi-structured data.

captured in natural language in the table. As shown in Figure 1 strings in cells often contain numerical values implicitly contextualized by surrounding text like "aged".

Current TQA systems (Xie et al., 2022b; Jiang et al., 2022; Liu et al., 2021a) linearize tables as a token string and process it jointly with the question. In response to long tables present in WikiTableQuestions (Pasupat and Liang, 2015a) and WikiSQL (Zhong et al., 2017) language models for TQA increase the context size to 1024 tokens incurring a high memory cost. However, we show in Section 3 as table size increases average model performance significantly degrades by 40%. Moreover, 23% of tables in WikiTableQuestions are truncated at 1024 tokens.

We propose ToolWriter , a new method that augments language model capabilities. In response to a question over a table, ToolWriter decides if tool use is required. It then generates a program to transform the table to simplify it to make question answering more effective. The generated tools are code that can be applied to all kinds of tables regardless of size which overcomes key language model limitations. The proposed ToolWriter approach is model agnostic and can be flexibly combined with existing models in a zero-shot setup.

In this work, we compare multiple detection strategies and tool generation approaches to generate query and table-specific Python programs. ToolWriter leverages our best combination for tool generation (zero-shot GPT-3) and tool-use detection (combined answer confidence and table length). Our method improves the state-of-the-art exact match results on WikiTableQuestions to 64.9 (+1.9%) and WikiSQL to 90.5 (1.5%).

We summarize our contributions as the following:

- We characterize the behavior of language models for TQA on WikiTableQuestions and find significant performance degradation as table size increases.
- We propose ToolWriter to detect when tool-use is required and generate query and table-specific programs as tools to transform tabular data. By generating row-filter tools we achieve new state-of-the-art results on WikiTableQuestions and WikiSQL.
- Through ablation studies analysis we show all our tool generation methods are model agnostic and improve effectiveness as table size increases.

2 Task Definition

Tabular question answering is a task in natural language processing that involves leveraging information from a semi-structured table T to generate an answer $\hat{y}$ to a question q. Questions are expressed in natural language and implicitly involve compositional types of reasoning to access and aggregate information in the table. These questions are implicitly multi-step and require a combination of symbolic reasoning and natural language understanding.

A system has access to a training set $D = \{(x = (q, T), \hat{y})\}$ of questions, tables, and answers. Tables between training and evaluation are disjoint to prevent memorization. The only restriction on the question is that it must be answerable given the information provided in the table. Average exact match (EM) over D between the predicted answer $\hat{y}$ and target y is used as the primary metric.

2.1 Datasets

WikiTableQuestions (Pasupat and Liang, 2015b) serves as our initial exploration into the limitations of current models. It is a tabular question-answering dataset from 2,108 HTML tables and crowdsourced question-answer pairs. Despite multiple questions per table in both train and test settings, tables between the training and testing set are distinct. WikiTableQuestions boasts several key attributes that make it an effective and challenging benchmark:

- Questions often require multiple steps to answer by gathering distinct pieces of information from a single table.
- Tables are not perfectly formatted often displaying non-consistent cell values depending on the implicit capabilities of the reader to discern different sections.
- Cells often contain interleaved formal representations and natural language making use of pure programmatic approaches challenging.

WikiTableQuestions-Filter is a subset of the WikiTableQuestions dev set for analysis in Section 5.1 to isolate tool performance in ToolWriter independent from the detector. Leveraging SQUALL annotations (Shi et al., 2020) we keep samples that contain a `WHERE` clause after a `SELECT`. This results in 1256 question-table-answer triplets.

WikiSQL (Zhong et al., 2017), similar to WikiTableQuestions, consists of 80,654 question-answer pairs over 24,241 tables from Wikipedia. Although its original intention was for semantic parsing it has been adapted to weak supervision settings by just using the target answer span as the source of signal. All tables in the dataset are fully parseable with types. Questions are simpler compared to WikiTableQuestions and only contain operations on full cell values that are fully parseable by an SQL query.

2.2 Baseline models

We investigate the limitations systems with varying task specific supervision.

BART (Lewis et al., 2020) is a Transformer (Vaswani et al., 2017) pre-trained with a denoising objective. For TQA it is fine-tuned with 1024 tokens of context jointly processing the query and a linearized table as follows: $x = q\,\texttt{[HEAD]}, c1, , cN, \texttt{[ROW]}, 1, r1, \texttt{[ROW]}, 2, r2..$

TapEx (Liu et al., 2021b) is a BART model fine-tuned to mimic an SQL executor on 5 million grammar-generated SQL statements. TapEx is currently state-of-the-art on the weak formulation of WikiSQL.

Omnitab (Jiang et al., 2022) is based on TapEx and further fine-tuned on natural language. The pre-training translates synthetic SQL queries into questions and mines similar passages to tables for masked language modeling. OmniTab is state-of-the-art on WikiTableQuestions and may be seen as the narrowest model for TQA due to its fine-tuning regime.

UnifiedSKG Xie et al. (2022a) is a T5 transformer (Raffel et al., 2020) with a standardized multi-task text-to-text format on structured knowledge (tables, knowledge bases, semantic parsing, etc...).

FlanT5 (Wei et al., 2021) takes a middle-ground approach between strong supervision and generality by instruction-tuning transformers on 62 different types of NLP tasks. Through in-context learning, it provides a strong baseline for TQA.

GPT-3 (Brown et al., 2020) showcases an unsupervised in-context learning approach to TQA. GPT-3 shows strong performance in TQA with zero-shot Chain-of-Thought (CoT) reasoning explicitly answering step-by-step (Kojima et al., 2022; Wei et al., 2022; Chen, 2022).

3 Behaviour Analysis

Tabular question answering is a challenging setting since tables can be exceedingly long. Context length in Transformer architectures is often limited to 512 tokens due to a quadratic memory cost (Vaswani et al., 2017). In WikiTableQuestions 41.7% of linearised tables exceed 512 tokens without even considering the question tokens. Current approaches to TQA patch this problem by increasing the context limit to 1024 tokens (Omnitab, TapEx, UnifiedSKG) and 2048 in GPT-3. This incurs a significant memory cost often prohibiting the use of such models. However, even at 1024 tokens of context 23.8% of tables are truncated thus incurring data loss.

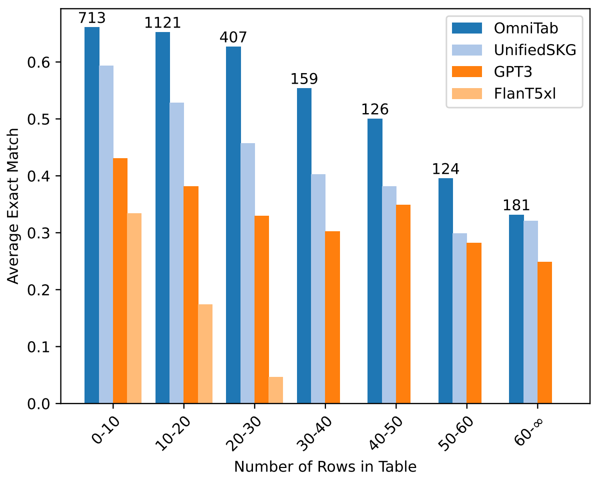


Figure 2: Exact match by table size on the WikiTableQuestions dev set. The number of dataset samples per row subset is shown above each bar.

3.1 Effectiveness across table size

Figure 2 shows model effectiveness stratified by the number of rows in a table on partitions of WikiTableQuestions dev. We note that the 1024 token context window is often exceeded at 40-50 rows. Interestingly, we observe a universal degradation in performance well before 40 rows. As such, table size is an important factor in performance independent of model capacity.

For tables exceeding 50 rows model performance decreases by an average of 40% relative to small tables. In these cases, due to prior ordering of the table, some questions only require information that is located at the top of the table. This maintains base performance for most models however we observe FlanT5-XL significantly degrades. Here we qualitatively observe hallucinations that repeat the input table tokens with large tables.

3.2 Potential in row filtering

Following our findings that table length has a significant and universal decrease in performance, we test the effect of filtering noise from the table. We hypothesize that removing noise from the data will increase model performance. We manually simulate a row-filtering tool by removing noise and only keeping the rows that are sufficient to answer the question.

To test our hypothesis we manually annotate 51 samples of the WikiTableQuestions dev set. We validate that correct usage of a tool is critical by also simulating a random row filter that removes 50% of rows indiscriminately. We note that the

Row Filter	U.SKG	O.Tab	GPT-3	FlanT5-XL
None	39.21	54.9	47.05	17.64
Random	11.76	5.88	15.64	5.52
Manual	**60.78**	**58.82**	**54.9**	**39.21**

Table 1: Exact match scores over 51 samples from the WikiTableQuestions dev set with gold rows selected manually as sufficient to answer the question.

format for the input to the models is kept constant as we only change the number of rows in a table. Our findings are shown in Table 1 pertaining to correct and incorrect usage of a row filtering tool where we draw two conclusions:

Correct row-filtering markedly improves performance across all tested model types. As tables reduce in size but retain relevant information, 4 of the 51 tables that are originally truncated to the 1024 token limit are fully processed. Noise is reduced by changing incorrect answers to positive ones in 10% of samples.

Incorrect usage of a tool poses the risk of removing relevant information. Since information in TQA is discreetly stored in table cells, it is challenging for models to recover once it is removed as we see by random row-filter performance. Furthermore, 7 of the 51 tables require no filtering. For such questions like *"how many players participated?"* we observe that row-filtering is query-specific and demands detection strategies.

Our findings motivate an automatic appraoch to generate a query-specific row-filtering tool and to detect when to apply it.

4 ToolWriter

ToolWriter is our proposed method to address the limitations of current language models on large semi-structured data. In response to a question over a table, ToolWriter decides if tool-use is required. It then generates a program to transform and simplify the table to make question-answering more effective. First, we outline our conceptual framework followed by our method implementation.

4.1 Proposed model

We introduce ToolWriter as TW that combines a $Tool$ with a TQA model F mediated by $\sigma \in [0, 1]$ where 0 means "don't apply tool" and 1 means "apply tool".

$$TW(x, F) = \sigma \cdot F(Tool(x)) + (1 - \sigma) \cdot F(x) \tag{1}$$

Moreover, we define our $sigma$ as the output of a *detector* function that aims to approximate the uncertainty of the model prediction.

$$\sigma = d_\theta(x, F(x)) \approx P\left(y \neq F(x)\right) \tag{2}$$

Our task is to maximize the exact match EM over our corpus D. We now take the partitions of D over the correct and wrong predictions of the model F (Eq. 3).

$$\begin{aligned} \overline{S_F} &= \{(x, \hat{y}) \in D | F(x) \neq \hat{y}\} \\ S_F &= \{(x, \hat{y}) \in D | F(x) = \hat{y}\} \end{aligned} \tag{3}$$

As we intend, transforming an input x with a tool might have a positive or negative effect on the produced output. When we observe our two subsets of D we can draw the following conclusion.

$$EM(TW(\cdot, F), \overline{S_F}) = \begin{cases} 0 & \sigma = 0 \\ \leq 1 & \sigma = 1 \end{cases} \tag{4}$$

$$EM(TW(\cdot, F), S_F) = \begin{cases} 1 & \sigma = 0 \\ \leq 1 & \sigma = 1 \end{cases} \tag{5}$$

The transformation on x produced by the tool is guaranteed to increase or maintain performance when the model is known to be wrong (Eq 4) and decrease it otherwise (Eq 5). This justifies our choice in Eq 2 of a detector that approximates the probability for an incorrect prediction of model F.

4.1.1 Tool use detection

In an ideal setting, we maximize EM over all available subsets: S_F and $\overline{S_F}$. σ mediates when we decide to use a tool. For S_F we can see the best choice is to use the original prediction yet for $\overline{S_F}$ we stand to gain if we use our tool before calling the model F. If we are in an oracle setting and we know the ground truth answer this gives us an oracle detector for when to use a tool.

$$\sigma = \mathbb{1}(y \neq F(x)) \tag{6}$$

However, in practice we approximate the oracle detector with a parametrized detector d. This aligns with previous work about query performance prediction (Cronen-Townsend et al., 2002) where estimating query difficulty is a reasonable assumption.

We explain our parametrized detector in more detail in Section 4.2. In the extreme, however, $\sigma = 1$ equates to always applying the tool. Given that we are detecting when the model is likely to fail, how we define our tool directly impacts the performance we can have on the subset of D it is applied over.

4.1.2 Programs as tools

The subset $\overline{S_F}$ is by definition difficult for F. As we see later in Section 3, our tested diverse set of state-of-the-art neural models (each corresponding to a specific instantiation of F) display similar patterns. This raises the natural following question: **What tool can distinctly complement the learned abilities of neural models?**

Our working hypothesis is that programs: 1) provide a natural interface to structured data; 2) circumvent several innate limitations of current neural systems due to their extrapolative nature. We define tools as short programs to transform the input data x into x^*, which are in the same input domain I. The term ϵ is introduced to account for any noise introduced between the transformed input (x^*) and the original input domain (I). This is represented in Eq 7 as:

$$Tool : I \rightarrow I^* + \epsilon \tag{7}$$

When paired selectively with an effective detection strategy d, programs as tools are applied to increase the likelihood of a correct prediction on a subset that is challenging for a model F. This outlines a program generation method $C(x)$ that generates a Python code dependent on the input. In Section 4.2 we describe our exploration into various methods for generating short programs as tools to interface in a query-specific way with structured data.

$$Tool(x) = Exec[x, C(x)] \tag{8}$$

Deciding which programs to create as tools are strongly correlated with the target task as well as the limitations of the models. As such, we first clearly define our task in Section 2 followed by an in-depth analysis of where our search for useful programs as tools will start.

4.2 Model implementation

ToolWriter is composed of a tool use detector and a query-specific tool generator on top of an existing TQA model. Following our behavior analysis, OmniTab and UnifiedSKG act as our two best TQA models $F(x)$ for WikiTableQuestions and we use TapEx for WikiSQL.

4.2.1 Model agnostic tool-use detector

We develop a model-agnostic detector $d_\theta(x, F(x))$ to detect when a tool is likely to improve model accuracy. Detecting input difficulty is a reasonable assumption and aligns with previous work on quality estimation (Ueffing and Ney, 2005; Fomicheva et al., 2020) and query performance prediction (Cronen-Townsend et al., 2002). The **combined** detector in ToolWriter is a linear classifier with the following features:

Sequence log-probability (SeqLogProb) is the length-normalised sequence log-probability from a trained model $F(y \mid y, x, \theta)$.

$$\frac{1}{L}\sum_{k=1}^{L} \log F(y_k \mid y_{<k}, x, \theta) \tag{9}$$

We expect low-confidence answers are likely to be incorrect.

Input length, as we have seen in Section 3, poses a challenge to all models irrespective of size and training objective. We leverage the size of the table measured by the number of rows as a simple feature to decide when to apply a tool.

Our use of such simple detection methods contrasts with well-studied error detection methods in NLP (Bérard et al., 2019; Fomicheva et al., 2020) as a sign that tools are reasonable model-agnostic extensions even with simple detection heuristics.

4.2.2 Row-filter tool generator

The tool generator synthesizes a short Python program that takes a table as input and returns a transformed version of it. The following code snippet is fixed and highlights the area where the generated code from the tool generator is placed.

```
new_table = table[table.apply(lambda ... , axis=1)]
```

The task of the row filter generator is to generate a lambda function to remove the rows in the table that are not relevant to answering the question. The

```
# Extracts x from “x-y” and keep if greater than 2
lambda row: float(row['Score'].split('–')[0]) >= 2
```

(a) Zero-shot GPT-3 for the question "In how many games did sri lanka score at least 2 goals?"

```
# Keep rows containing ‘France’ in the description
lambda row: 'france' in row['Description'].lower()
```

(b) T5 for the question "Is France mentioned positively?"

Figure 3: Examples row filter tools generated from our two proposed methods. Comments are added manually for explanatory purposes.

Tool	Detector	Omnitab	UnifiedSKG
Baseline		73.5	55.5
T5	**Always**	72.0 (-1.4)	59.5 (+4.0)
	Oracle	77.5 (+4.0)	63.3 (+7.8)
GPT-3	**Always**	74.6 (+1.1)	63.0 (+7.6)
	Oracle	80.3 (+6.8)	68.6 (+13.1)
Human SQL	**Always**	74.9 (+1.4)	65.0 (+9.5)
	Oracle	82.7 (+9.2)	70.6 (+15.1)

Table 2: Row filter tool performance on WikiTableQuestions-Filtered with two detection strategies.

program may be of arbitrary complexity emphasizing the generality of our approach for systems to interact with data through programs as seen in Figure 3.

As we see in Table 1 removing rows requires care to preserve crucial information. It is evident that a tool must adapt its filtering strategy according to the question and the table. Our row filtering tool keeps its input and output space consistent and suitable for the downstream model.

The task of generating tools requires the model to produce an explicit transformation of the table given the question. Given that the search space for tools as programs grows exponentially with the expressiveness of the tools, we opt for methods that reduce the search space by having a prior on what possible transformations will work best.

Specifically, we explore 2 approaches for generating Python row filters:

Fine-tuned T5. We fine-tune a T5 model through supervised training to autoregressively generate a Python row filter given a question q and a table T. For our supervised data we leverage a subset WikiTableQuestions with SQUALL (Shi et al., 2020) annotations on questions that contain a single `SELECT` and `WHERE` clause which are likely to benefit from row filtering. Our starting checkpoint is FlanT5-XL, using a batch size of 64 on two RTX 3090 GPUs for 10k steps for 8 hours. Further training details are in

Zero-shot GPT-3. We leverage GPT-3 for zero-shot prediction to generate a row filter as a Python lambda function. We use the "text-davinci-003" API with a temperature of 0.2 with the question and table schema in the prompt (Appendix C). Zero-shot tool generation shows the potential in low-effort approaches to manipulate structured data.

Figure 3 showcases generated Python code samples from both our proposed tool generators. Language models have no formal guarantees for executable code (Rae et al., 2021; Chen et al., 2021). As a result, if the execution of the tool throws an exception or the resulting table is empty we revert to the original table.

Manual tool. For WikiTableQuestions-Filter we leverage the SQUALL SQL annotations to derive a manual row filter. We analyze row filter headroom performance in Section 5.1.

5 Results

First, we investigate the various tool generators of ToolWriter on a subset of WikiTableQuestions where filtering is often required. Second, we focus on the importance of the detector to choose when to best apply the generated tools (Section 5.2). Third, we test ToolWriter to both detect and generate tools across various TQA datasets and methods (Section 5.3). Finally, we analyze how ToolWriter performs as table size increases (Section 5.4).

5.1 Performance of tool generators

As we observe in Table 2 our automatic tool generators (T5, zero-shot GPT-3) almost universally increase model performance on WikiTableQuestions-Filtered. Importantly, regardless of how our tools may be applied, UnifiedSKG significantly benefits by all tools generated by ToolWriter . This shows our tools are effective at filtering irrelevant information from tables that would otherwise cause TQA models to fail.

Manual tools show the potential for tool generators to simplify tables further. Our best tool generator, zero-shot GPT-3, achieves 70% of manual performance averaged over all detection settings and models.

Table 2 also informs us of the importance of the detector. We observe a large gap for all tool

	WikiTableQuestions				WikiSQL	
	OmniTab		UnifiedSKG		TapEx	
Detector	Dev	Test	Dev	Test	Dev	Test
Never apply	62.7	63.0	49.6	50.8	89.6	89.0
Always apply	56.5 (-6.2)	57.5 (-5.5)	48.5 (-1.0)	50.2 (-0.6)	89.6 (0.0)	89.8 (+0.7)
SeqLogProb	63.7 (+1.0)	64.3 (+1.2)	52.6 (+3.0)	54.6 (+3.8)	90.5 (+0.8)	90.4 (+1.3)
Combined	63.7 (+1.0)	64.9 (+1.8)	52.9 (+3.4)	54.5 (+3.7)	90.7 (+1.1)	90.5 (+1.5)
Oracle	67.4 (+4.7)	68.3 (+5.3)	57.9 (+8.3)	59.0 (+8.1)	91.7 (+2.0)	91.5 (+2.4)

Table 3: Exact match results on various detection strategies for applying our best row-filter tool generator: GPT-3.

generators comparing always applying our tool to oracle detection.

5.2 Detecting when to use tools

Table 3 shows performance of multiple detection strategies on the full dev and test sets for WikiTableQuestions and WikiSQL. We use our best-performing tool generator, zero-shot GPT-3. We observe row filtering tools require query specific detection since "always" or "never" applying tools shows the lowest results in all cases.

We observe that even simple detection methods like SeqLogProb are sufficient to inform ToolWriter when to apply the query-specific generated row filter. We see significant benefits in leveraging tools for all TQA models in contrast to not using them. Performance increases further as we include table length as a feature in our detector highlighting the importance of using tools in accordance with the complexity of the data.

Under oracle detection conditions we observe significant potential for our generated tools. This shows how deciding **when** to apply a row filter tool is just as important as **how** to apply it.

5.3 Overall performance

Leveraging our prior findings, ToolWriter is the combination of our best detection method (SeqLogProb with table length) and our best row-filter tool generator (zero-shot GPT-3). For each dataset, we show the corresponding model F as our base TQA model. Table 4 and Table 5 show overall model performance on WikiTableQuestions and WikiSQL respectively.

Our results show ToolWriter significantly improves performance agnostic of the target model using the generated tools. UnifiedSKG is particularly effective in leveraging the transformed tables with a 3.6% absolute performance increase compared to not using tools. When paired with Om-

Method	**Dev**	**Test**
2-shot GPT-3 Direct (Chen, 2022)	—	27.3
BART (Lewis et al., 2020)	37.2	38.0
2-shot GPT-3 CoT (Chen, 2022)	—	45.7
UnifiedSKG (Xie et al., 2022b)	50.9 (49.6)	50.9 (50.8)
ToolWriter + UnifiedSKG	**52.9**	**54.5**
TapEx (Liu et al., 2021a)	57.0	57.5
OmniTab (Jiang et al., 2022)	— (62.7)	62.8 (63.0)
ToolWriter + Omnitab	**63.7**	**64.9**

Table 4: Exact match accuracy results on WikiTableQuestions. Results in parenthesis are our reproduced experiments.

Method	**Dev**	**Test**
BART (Lewis et al., 2020)	87.3	85.8
UnifiedSKG (Xie et al., 2022b)	87.4	85.7
OmniTab (Jiang et al., 2022)	—	88.7
TapEx (Liu et al., 2021a)	89.2 (89.6)	89.5 (89.0)
ToolWriter + TapEx	**90.7**	**90.5**

Table 5: Exact match accuracy results on WikiSQL. Results in parenthesis are our reproduced experiments.

niTab and TapEx we improve the state-of-the-art for both datasets. The improvement in WikiSQL is particularly impactful as ToolWriter enables a 10% error-rate reduction.

These results show how programmatic tools effectively complement neural components as an effective method for processing semi-structured data. In the following section, we perform a stratified analysis to understand where ToolWriter leads to the most improvements.

5.4 Tools improve performance on long tables

In this section we do an ablation study stratified by table length on WikiTableQuestions: short tables

		Omnitab		UnifiedSKG	
		Dev	Test	Dev	Test
rows < 30	**Baseline**	67.0	67.9	52.7	54.3
	ToolWriter	67.0 (+0.0)	**68.0 (+0.1)**	**55.0 (+2.3)**	**57.2 (+2.8)**
$30 \leq$ rows < 60	**Baseline**	48.7	46.4	39.6	37.5
	ToolWriter	**52.6 (+3.9)**	**51.9 (+5.5)**	**44.0 (+4.4)**	**45.1 (+7.6)**
rows ≤ 60	**Baseline**	41.4	35.0	33.7	34.3
	ToolWriter	**48.1 (+6.6)**	**43.3 (+8.3)**	**42.0 (+8.3)**	**41.7 (+7.5)**

Table 6: Row filtering performance comparison on partitions stratified by table length for WikiTableQuestions.

(rows < 30), medium tables ($30 \leq$ rows < 60), and long tables ($60 \leq$ rows). We aim to quantify the effect ToolWriter has as table size increases. As in Section 5.3 ToolWriter uses GPT-3 as the tool generator and the combined detector.

Table 6 shows our original hypothesis confirmed: Row filtering tools can be an effective strategy to help models handle long tables. We notice how as table length increases, the positive effect of the row filtering tool becomes more pronounced. Our hypothesis is further confirmed with our T5 tool generator where results mimic the Table 6 reaching up to 5% absolute improvement with UnifiedSKG.

As noted in Section 4.2.1, detection is critical to tool-use. On short tables, we observe no degradation in performance highlighting the effectiveness of our combined detector.

6 Background and Related Work

Semantic parsing focuses on generating an executable parse for the exact answer (McClelland and Rumelhart, 1986), benefiting from data size independence (Herzig and Berant, 2017). It requires strong supervision (Dong and Lapata, 2018; Yin et al., 2021) or reinforcement learning (Zhong et al., 2017) and assumes coherent data formatting and an expressive target language.

Alternative approaches learn a joint table-question-answer mapping. Seq2Seq models (Sutskever et al., 2014) execute (Zaremba and Sutskever, 2014) and simulate formal programs (Lu et al., 2015). Intermediate executable modules were integrated (Neelakantan et al., 2015), while Transformer-based models (Vaswani et al., 2017; Lewis et al., 2020; Raffel et al., 2020) leveraged unsupervised language capabilities (Xie et al., 2022a; Jiang et al., 2022; Herzig et al., 2020; Yin et al., 2020).

Recent interest in execution-loop models arises from language models' ability to explain reasoning (Wei et al., 2022), improving compositional questions (Zhou et al., 2022) and symbolic manipulation (Bueno et al., 2022; Nye et al., 2021; Wolfson et al., 2020). TQA language models generate chains of thought with sub-question answers (Chen, 2022).

Recent advances in code-focused language models led to an interest in combining question decomposition and program interaction (Chen et al., 2021). Toolformer (Schick et al., 2023), Program Assisted Learning (Gao et al., 2022b), and Tool Augmented Language Models (Parisi et al., 2022) interleave execution and natural language reasoning but face limitations in capacity. Our work addresses large structured context directly while interleaving execution and natural language.

7 Conclusion

Tabular question answering is a challenging setting for neural methods due to large context sizes and implicit reasoning. First, we characterize the limitations of neural methods to integrate structured data and find all language modeling methods struggle with large tables. Second, we propose ToolWriter to generate query-specific tools to simplify large tables and detect when these transformations should be applied. We propose various language model-based methods to generate programs that filter rows which universally improve and achieve state-of-the-art results on two tabular question-answering datasets. Finally, we determine significant headroom in both detecting when to use tools and how to generate them under oracle setting highlighting the potential in tools to manipulate structured data combined with language models.

References

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, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff 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. *ArXiv*, abs/2005.14165.

Mirelle Candida Bueno, Carlos Gemmel, Jeffrey Stephen Dalton, Roberto de Alencar Lotufo, and Rodrigo Nogueira. 2022. Induced natural language rationales and interleaved markup tokens enable extrapolation in large language models. *ArXiv*, abs/2208.11445.

Alexandre Bérard, Ioan Calapodescu, and Claude Roux. 2019. Naver labs europe's systems for the wmt19 machine translation robustness task.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code.

Wenhu Chen. 2022. Large language models are few(1)-shot table reasoners. *ArXiv*, abs/2210.06710.

Stephen Cronen-Townsend, Yun Zhou, and W. Bruce Croft. 2002. Predicting query performance.

Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. In *Annual Meeting of the Association for Computational Linguistics*.

Marina Fomicheva, Shuo Sun, Lisa Yankovskaya, Frédéric Blain, Francisco Guzmán, Mark Fishel, Nikolaos Aletras, Vishrav Chaudhary, and Lucia Specia. 2020. Unsupervised quality estimation for neural machine translation.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022a. Pal: Program-aided language models.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022b. Pal: Program-aided language models. *ArXiv*, abs/2211.10435.

Jonathan Herzig and Jonathan Berant. 2017. Neural semantic parsing over multiple knowledge-bases. In *Annual Meeting of the Association for Computational Linguistics*.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. Tapas: Weakly supervised table parsing via pre-training. *ArXiv*, abs/2004.02349.

Zhengbao Jiang, Yi Mao, Pengcheng He, Graham Neubig, and Weizhu Chen. 2022. Omnitab: Pretraining with natural and synthetic data for few-shot table-based question answering. In *North American Chapter of the Association for Computational Linguistics*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners.

Angeliki Lazaridou, Elena Gribovskaya, Wojciech Stokowiec, and Nikolai Grigorev. 2022. Internet-augmented language models through few-shot prompting for open-domain question answering. *ArXiv*, abs/2203.05115.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension.

Qian Liu, Bei Chen, Jiaqi Guo, Zeqi Lin, and Jian-Guang Lou. 2021a. Tapex: Table pre-training via learning a neural sql executor. *ArXiv*, abs/2107.07653.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2021b. Tapex: Table pre-training via learning a neural sql executor.

Zhengdong Lu, Hang Li, and Ben Kao. 2015. Neural enquirer: Learning to query tables in natural language. In *IEEE Data Engineering Bulletin*.

James L. McClelland and David E. Rumelhart. 1986. Mechanisms of sentence processing: Assigning roles to constituents of sentences.

Arvind Neelakantan, Quoc V. Le, and Ilya Sutskever. 2015. Neural programmer: Inducing latent programs with gradient descent. *CoRR*, abs/1511.04834.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. 2021. Show your work: Scratchpads for intermediate computation with language models.

Aaron Parisi, Yao Zhao, and Noah Fiedel. 2022. Talm: Tool augmented language models.

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

Panupong Pasupat and Percy Liang. 2015b. Compositional semantic parsing on semi-structured tables.

Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, George van den Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, Amelia Glaese, Johannes Welbl, Sumanth Dathathri, Saffron Huang, Jonathan Uesato, John Mellor, Irina Higgins, Antonia Creswell, Nat McAleese, Amy Wu, Erich Elsen, Siddhant Jayakumar, Elena Buchatskaya, David Budden, Esme Sutherland, Karen Simonyan, Michela Paganini, Laurent Sifre, Lena Martens, Xiang Lorraine Li, Adhiguna Kuncoro, Aida Nematzadeh, Elena Gribovskaya, Domenic Donato, Angeliki Lazaridou, Arthur Mensch, Jean-Baptiste Lespiau, Maria Tsimpoukelli, Nikolai Grigorev, Doug Fritz, Thibault Sottiaux, Mantas Pajarskas, Toby Pohlen, Zhitao Gong, Daniel Toyama, Cyprien de Masson d'Autume, Yujia Li, Tayfun Terzi, Vladimir Mikulik, Igor Babuschkin, Aidan Clark, Diego de Las Casas, Aurelia Guy, Chris Jones, James Bradbury, Matthew Johnson, Blake Hechtman, Laura Weidinger, Iason Gabriel, William Isaac, Ed Lockhart, Simon Osindero, Laura Rimell, Chris Dyer, Oriol Vinyals, Kareem Ayoub, Jeff Stanway, Lorrayne Bennett, Demis Hassabis, Koray Kavukcuoglu, and Geoffrey Irving. 2021. Scaling language models: Methods, analysis & insights from training gopher.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li 0133, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *ArXiv*, abs/2302.04761.

Tianze Shi, Chen Zhao, Jordan Boyd-Graber, Hal Daumé, and Lillian Lee. 2020. On the potential of lexico-logical alignments for semantic parsing to sql queries.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*.

Nicola Ueffing and Hermann Ney. 2005. Word-level confidence estimation for machine translation using phrase-based translation models.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2019. Huggingface's transformers: State-of-the-art natural language processing.

Tomer Wolfson, Mor Geva, Ankit Gupta, Matt Gardner, Yoav Goldberg, Daniel Deutch, and Jonathan Berant. 2020. Break it down: A question understanding benchmark. *Transactions of the Association for Computational Linguistics*, 8:183–198.

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. 2022a. Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models.

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 R. Radev, Caiming Xiong, Lingpeng Kong, Rui Zhang, Noah A. Smith, Luke Zettlemoyer, and Tao Yu. 2022b. Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. In *Conference on Empirical Methods in Natural Language Processing*.

Pengcheng Yin, Hao Fang, Graham Neubig, Adam Pauls, Emmanouil Antonios Platanios, Yu Su, Sam Thomson, and Jacob Andreas. 2021. Compositional generalization for neural semantic parsing via span-level supervised attention. In *North American Chapter of the Association for Computational Linguistics*.

Pengcheng Yin, Graham Neubig, Wen tau Yih, and Sebastian Riedel. 2020. Tabert: Pretraining for joint understanding of textual and tabular data. *ArXiv*, abs/2005.08314.

Wojciech Zaremba and Ilya Sutskever. 2014. Learning to execute. *ArXiv*, abs/1410.4615.

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

Denny Zhou, Nathanael Scharli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Olivier Bousquet, Quoc Le, and Ed Huai hsin Chi. 2022. Least-to-most prompting enables complex reasoning in large language models. *ArXiv*, abs/2205.10625.

A Experimental setting

We use the recommended settings for each and linearize the tables with the official scripts. At test time we perform greedy decoding. We obtain these models from the Huggingface model hub (Wolf et al., 2019).

Our two tool generation models: prompt-based and fine-tuned. For our prompt-based model we use the OpenAI GPT-3 API with the "text-davinci-003" model with a temperature of 0.2.

Our fine-tuned FlanT5-XL model is always trained for 10k steps with the validation set used to tune the appropriate batch size of 64, weight decay of 0.01 and a learning rate of 10e-4. We perform a grid search across these parameters as the most influential for the final row filter effectiveness.

B Subset analysis

To effectively filter according to relevant criteria we leverage parallel SQL annotations from SQUALL (Shi et al., 2020) which cover 77.11% of the dev data. These annotations are formal semantic parses of the query which remove the natural language variability enabling us to filter by required capability.

As shown in the first two rows of Table 7 we see a marked decrease in performance for non SQUALL annotated samples across all model types. These are cases where the queries or table are considered too complex to be expressed as SQL. As such we are restricted to qualitative and point-wise analysis of these samples to characterize model behavior.

C Prompts

Data Subset	Uni.SKG	OmniTab	GPT-3	FlanT5	dataset %
SQUALL annotated	54.56	66.65	38.62	17.36	77.11
Non SQUALL annotated	33.49	40.43	30.4	11.27	22.89
+ 1’ as offset	69.23	73.85	47.69	16.92	2.3
requires counting rows	43.14	66.11	27.17	6.72	25.22
count all rows	39.62	66.67	16.98	3.77	5.62
big sub or add	6.58	11.84	55.26	2.63	2.68
1 ’where’ clause	55.65	70.75	39.92	17.63	44.68
2 ’where’ clauses	49.42	59.83	41.62	15.32	12.22
count and where	54.41	67.8	39.73	16.35	63.3
answer not in prompt	30.23	31.4	26.74	3.88	9.11
multiple answers	41.94	51.61	32.26	0	2.19
one select	55.92	68.33	38.15	17.88	63.79
one select and one where	55.99	71.2	40.06	17.7	43.91
has duplicate columns	30.59	41.18	21.18	10.59	3

Table 7: Exact match performance of systems according to subsets of the WikiTableQuestions dev set.

```
User 1:
I need an expert to help me answer the question by making the table
smaller.
Question: Who are all of the players on the Westchester High School
club team?

table = {’Player’: [’Jarrett Jack’, ’Jermaine Jackson’, ...
’No.’: [’1’, ’8’, ...
’Nationality’: [’United States’, ’United States’, ...
’Position’: [’Guard’, ’Guard’, ...
’Years in Toronto’: [’2009-10’, ’2002-03’, ...
’School/Club Team’: [’Georgia Tech’, ’Detroit’, ...
}

User 2:
For ’Who are all of the players on the Westchester High School club
team?’ the most impactful change will be to filter the rows. Since I
don’t know all the rows I’ll use rough string matching, float casting,
lowering and be as broad as possible.

>>> new_table = table[table.apply(lambda row_dict:  ’Westchester’ in
row_dict[’School/Club Team’].lower(), axis=1)]
```

Figure 4: Prompt used to generate row filter tools with GPT-3 in a zero-shot setup. Tables are truncated to 2 rows to give the model a schema for how to interact with the data. Hilighted region indicates the start of the prompt completion.