Table-GPT: Table Fine-tuned GPT for Diverse Table Tasks

PENG LI*, Georgia Institute of Technology, USA
YEYE HE, Microsoft Research, USA
DROR YASHAR, Microsoft, Israel
WEIWEI CUI, Microsoft Research, China
SONG GE, Microsoft Research, China
HAIDONG ZHANG, Microsoft Research, China
DANIELLE RIFINSKI FAINMAN, Microsoft, Israel
DONGMEI ZHANG, Microsoft Research, China
SURAJIT CHAUDHURI, Microsoft Research, USA

Language models, such as GPT-3 and ChatGPT, demonstrate remarkable abilities to follow diverse human instructions and perform a wide range of tasks, using *instruction fine-tuning*. However, when we probe language models with a range of basic table-understanding tasks, we observe that today's language models are still sub-optimal in many table-related tasks, likely because they are pre-trained predominantly on *one-dimensional* natural-language texts, whereas relational tables are *two-dimensional* objects.

In this work, we propose a new "*table fine-tuning*" paradigm, where we continue to train/fine-tune language models like GPT-3.5 and ChatGPT, using diverse table-tasks synthesized from real tables as training data, which is analogous to "instruction fine-tuning", but with the goal of enhancing language models' ability to understand tables and perform table tasks. We show that our resulting Table-GPT models demonstrate: (1) better *table-understanding* capabilities, by consistently outperforming the vanilla untuned GPT-3.5 and ChatGPT, on a wide range of table tasks (data transformation, data cleaning, data imputation, table-QA, etc.), including tasks that are completely *holdout and unseen* during training, and (2) strong *generalizability*, in Table-GPT's ability to respond to diverse human instructions to perform *new and unseen* table-tasks, in a manner similar to GPT-3.5 and ChatGPT.

Our code, training data, as well as an extensive evaluation benchmark for table-tasks, are released at https://github.com/microsoft/Table-GPT for future research.

CCS Concepts: • **Information systems → Data management systems**.

Additional Key Words and Phrases: Language Models, Table Models, Table Fine-tuning, Instruction Fine-tuning, Multi-task Training, Table Tasks, Synthesized Training Data, Model Generalizability, Unseen Tasks

[*] Work done while at Microsoft.

Authors' addresses: Peng Li, Georgia Institute of Technology, Atlanta, USA, pengli@gatech.edu; Yeye He, Microsoft Research, Redmond, USA, yeyehe@microsoft.com; Dror Yashar, Microsoft, Redmond, Israel, dror.yashar@microsoft.com; Weiwei Cui, Microsoft Research, Redmond, China, weiweicu@microsoft.com; Song Ge, Microsoft Research, Redmond, China, songge@microsoft.com; Haidong Zhang, Microsoft Research, Redmond, China, haizhang@microsoft.com; Danielle Rifinski Fainman, Microsoft, Redmond, Israel, danielle.rifinski@microsoft.com; Dongmei Zhang, Microsoft Research, Redmond, China, dongmeiz@microsoft.com; Surajit Chaudhuri, Microsoft Research, Redmond, USA, surajitc@microsoft.com.

ACM 2836-6573/2024/6-ART176
https://doi.org/10.1145/3654979

ACM Reference Format:
Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. 2024. Table-GPT: Table Fine-tuned GPT for Diverse Table Tasks. *Proc. ACM Manag. Data* 2, 3 (SIGMOD), Article 176 (June 2024), 28 pages. https://doi.org/10.1145/3654979

1 INTRODUCTION

Large language models, such as GPT and LLaMA, have recently demonstrated impressive abilities in performing diverse natural-language tasks [8, 11, 15, 67]. Considering the abundance of table data (e.g., database tables and spreadsheet tables) and the value they represent, it is essential to study and improve language models' abilities on understand tables. In the literature, a number of pioneering works, such as [25, 38, 52, 56], have also shown that by using "*prompt engineering*", or carefully selecting the best instructions and few-shot examples for a particular task, language models can perform well on a number of table-tasks such as entity matching and data imputation.

While prompt-engineering is a promising direction to enhance model performance, it requires task-specific tuning (e.g., using task-specific labeled data to find the best instruction/example combinations to use in prompts) [9, 11, 81]. We in this work propose an orthogonal paradigm called "*table-tuning*", where instead of modifying prompts, we modify the weights of the underlying language models *for once* (i.e., not task-specific), by continuing to train the models using diverse table-tasks to improve their ability to understand tables (which is analogous to the use of instruction-tuning to improve models' ability to follow instructions [53, 76]). We show that table-tuned Table-GPT consistently outperforms the vanilla GPT-3.5 and ChatGPT on a wide range of table tasks (data transformation, data cleaning, data profiling, data imputation, table-QA, etc.), including *new and unseen* table-tasks not used in training. We note that our model-tuning approach is also *complementary to* prompt-engineering, because carefully engineered prompts can continue to benefit both vanilla language models and our table-tuned models.

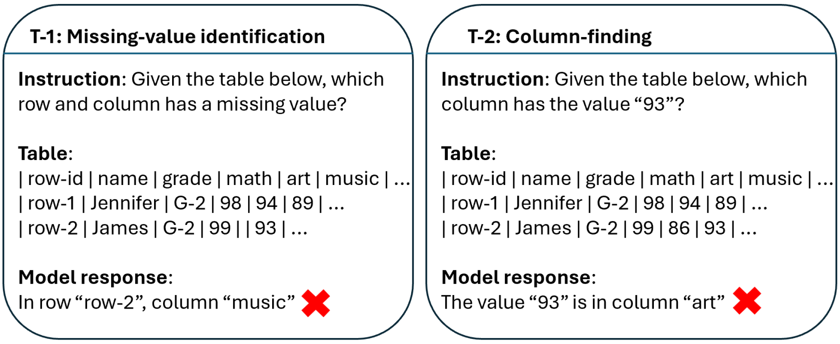
Today's language models cannot "read tables" reliably. While today's language models excel in natural-language tasks, we start by asking the question of whether these models are optimal for table-tasks, because after all, they are pre-trained predominantly on texts, which are different from tables in many ways.

Specifically, natural language texts are generally (1) *one-directional*, (2) read *left-to-right*, where (3) swapping two tokens will usually change the meaning of the text. In contrast, relational tables are (1) *two-dimensional* in nature with both rows and columns, where (2) reading *top-to-bottom* in the vertical direction (for values in the same column) is crucial in many common table-tasks. Furthermore, (3) unlike text, tables are largely "invariant" to row and column permutations, where swapping two rows or columns does not generally change the semantic meaning of a table.

With these in mind, we perform two simple tests to probe language models' ability to "read" tables and then answer basic questions, which we refer to as (T-1) Missing-value-identification, and (T-2) Column-finding, as shown in Figure 1.

In (T-1) Missing-value-identification, we show language models with a real table, presented in a markdown [2] or other formats [1], where we make sure that there is exactly one empty cell in the table. We then ask the models to identify the empty cell, by responding with the cell's column-name and row-id, repeating 1000 times using 1000 randomly sampled real tables. Despite the impressive ability of language-models like GPT-3.5 to perform diverse NLP tasks, we find that they fail on a surprisingly large fraction (up to 74%) of such tests, by responding with incorrect column-headers

[1] Markdown table is the format that models like GPT prefer to use when generating table responses, presumably they are pre-trained on GitHub data, where markdown-tables are abundant. We also test other table formats such as JSON and XML, which we will discuss later in the experiments.



T-1: Missing-value identification

Instruction: Given the table below, which row and column has a missing value?

Table:
| row-id | name | grade | math | art | music | ...
| row-1 | Jennifer | G-2 | 98 | 94 | 89 | ...
| row-2 | James | G-2 | 99 | | 93 | ...

Model response:
In row "row-2", column "music" ✗

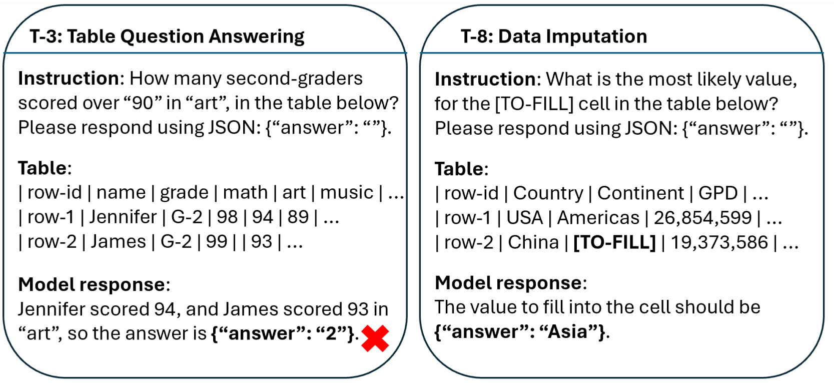
T-2: Column-finding

Instruction: Given the table below, which column has the value "93"?

Table:
| row-id | name | grade | math | art | music | ...
| row-1 | Jennifer | G-2 | 98 | 94 | 89 | ...
| row-2 | James | G-2 | 99 | 86 | 93 | ...

Model response:
The value "93" is in column "art" ✗

Fig. 1. Two simple tests to probe language-models' ability to read tables. (Left) T-1: Missing value identification, which is to identify the column-header/row-id of a missing cell. (Right) T-2: Column-Finding, which is to identify the column-name of a given value. Even large models (e.g. 175B GPT-3.5 models like text-davinci-002) can frequently fail on such tests, with only 0.26 accuracy in one variant of the tests.



T-3: Table Question Answering

Instruction: How many second-graders scored over "90" in "art", in the table below? Please respond using JSON: {"answer": ""}.

Table:
| row-id | name | grade | math | art | music | ...
| row-1 | Jennifer | G-2 | 98 | 94 | 89 | ...
| row-2 | James | G-2 | 99 | | 93 | ...

Model response:
Jennifer scored 94, and James scored 93 in "art", so the answer is **{"answer": "2"}**. ✗

T-8: Data Imputation

Instruction: What is the most likely value, for the [TO-FILL] cell in the table below? Please respond using JSON: {"answer": ""}.

Table:
| row-id | Country | Continent | GPD | ...
| row-1 | USA | Americas | 26,854,599 | ...
| row-2 | China | **[TO-FILL]** | 19,373,586 | ...

Model response:
The value to fill into the cell should be **{"answer": "Asia"}**.

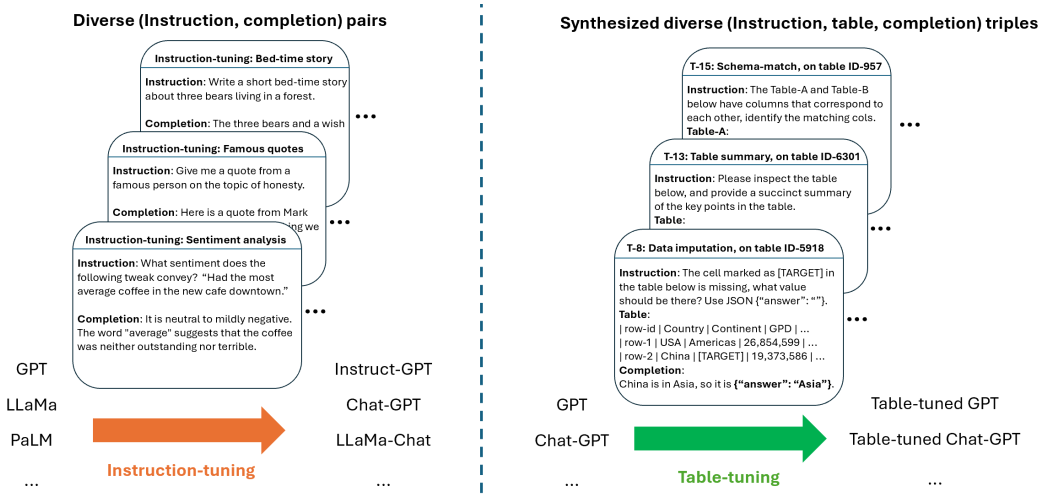
Fig. 2. Example table-tasks, where the ability of language models to "read" tables vertically is important. (Left) T-3: Table Question-Answering. (Right) T-8: Data Imputation. More tasks like these are shown in Table 2.

or row-ids – for instance, in the example shown in Figure 1, the model may answer that the missing cell is located in the column "`music`", when the correct answer should be "`art`".

In order to ensure that there is no ambiguity in what "missing value" or "empty cell" could mean to language models, we design a second and even simpler test, which we refer to as (T-2) Column-finding, shown on the right of Figure 1. In this test, we present a language model with a real table, and ask it to find a specific cell-value that appears exactly once in the entire table (e.g., "`93`" in this example), and then respond with the column-name of the that value. We find that language models such as GPT-3.5 are prone to fail on such tests again (e.g., answering that "`93`" is in the column "`art`" when the correct answer should be "`music`"), failing on over half of such tests.

These simple probes show that today's large language models, when pre-trained on large amounts of one-directional natural-language texts, are not best-suited to "read" two-dimensional tables, especially in the vertical direction, which however is crucial for many common table-tasks.

Consider, for example, the popular NLP task of (T-3) Table-QA [14, 55, 66], where the task is to answer a natural-language question, based on the content of a given table. The left side of Figure 2 shows such an example, where the question is "`How many second-graders scored over 90 in art, in the table below?`" Imagine that a model is not able to "read" tables correctly, it may believe


Diverse (Instruction, completion) pairs
Instruction-tuning: Bed-time story
Instruction-tuning: Famous quotes
Instruction-tuning: Sentiment analysis
GPT
LLaMa
PaLM
...
Instruction-tuning
Instruct-GPT
Chat-GPT
LLaMa-Chat
...
Synthesized diverse (Instruction, table, completion) triples
T-15: Schema-match, on table ID-957
T-13: Table summary, on table ID-6301
T-8: Data imputation, on table ID-5918
GPT
Chat-GPT
...
Table-tuning
Table-tuned GPT
Table-tuned Chat-GPT
...

Instruction: Write a short bed-time story about three bears living in a forest.

Completion: The three bears and a wish

Instruction: Give me a quote from a famous person on the topic of honesty.

Completion: Here is a quote from Mark

Instruction: What sentiment does the following tweak convey? “Had the most average coffee in the new cafe downtown.”

Completion: It is neutral to mildly negative. The word "average" suggests that the coffee was neither outstanding nor terrible.

Instruction: The Table-A and Table-B below have columns that correspond to each other, identify the matching cols.
Table-A:

Instruction: Please inspect the table below, and provide a succinct summary of the key points in the table.
Table:

Instruction: The cell marked as [TARGET] in the table below is missing, what value should be there? Use JSON {“answer”: “”}.
Table:
| row-id | Country | Continent | GPD | ...
| row-1 | USA | Americas | 26,854,599 | ...
| row-2 | China | [TARGET] | 19,373,586 | ...
Completion:
China is in Asia, so it is **{“answer”: “Asia”}**.

Fig. 3. Instruction-tuning vs. Table-tuning. (Left): Instruction-tuning is a technique developed in the NLP literature that continues to train language models (e.g., GPT) for instruction-following capabilities (e.g., leading to ChatGPT). (Right): Table-tuning is an analogous approach we propose to train language models to better understand tables and perform table-tasks.

that both “`Jennifer`” and “`James`” satisfy the condition (because it believes “`93`” is in the column “`art`”, as in Figure 1 (Right)), and therefore answer “`2`”, instead of the correct answer “`1`”.

We emphasize that the ability to read tables in the vertical direction (top-to-bottom for values in the same column), is similarly important in many other table-tasks, such as data-imputation (shown on the right of Figure 2), data-transformation, error-detection, and even code-related tasks such as NL-to-SQL (e.g., if a natural-language utterance such as “`93`” cannot be located in the correct column, then the code generated by language models to filter rows using the value “`93`”, may also be referencing incorrect columns). These are just examples of many other table-tasks we study in this work (listed in Table 2), where the ability to read tables correctly is crucial.

Furthermore, we find that large language models can be sensitive to the order in which rows and columns are presented in a table – e.g., when we swap the order of two rows and columns in a table, a model can change its response to a table-task, even when such a swap should not change the semantic meaning of the table. This is presumably because language-models are pre-trained on text where the order of tokens matters (e.g., “`Jennifer called you`” vs. “`you called Jennifer`”), leading to sub-optimal behaviors when tables are used as input (which should generally be invariant to row and column permutations).

There are more observations like these, and we believe they all point to new research opportunities for us to improve the underlying language model, by enhancing their ability to understand tables and perform table-tasks.

Instruction-tuning in NLP: train language models to follow diverse human instructions. To change the behaviour of language models, successful attempts have been made in the NLP community, using a technique known in the literature as “instruction-tuning” [53, 61, 76–78].

It was observed in the NLP literature [11, 53, 78], that while early versions of pre-trained language models are able to predict the likely next token (e.g., “`write a bedtime`” ⟶ “`story`”), they cannot reliably follow higher-level instructions from humans (e.g., “`write a bedtime story about a bear, for a 3 years old, in 100 words`”) – the latter is a behavior only demonstrated in later models like ChatGPT, through the instruction-tuning process, as illustrated on the left of Figure 3.

Specifically, in instruction-tuning, diverse training data in the form of "`(instruction, completion)`" pairs are constructed, often manually annotated by human labellers [53], e.g. ("`write a bedtime story about a bear`" → `an-actual-human-written-story`), as demonstrations for language-models to learn how to follow high-level human instructions. Such data are then used to continue to train language models, to improve their ability to understand and follow instructions, leading to well-known models such as ChatGPT/InstructGPT [3, 53], as well as their open-source counterparts, like Stanford-Alpaca [5] and LLaMA-chat [67].

Table-tuning: train language models to understand tables. We believe that the success of the instruction-tuning research in NLP holds lessons for us, when we aim to enhance language models' ability to understand tables.

In this work, we propose a "*table-tuning*" paradigm analogous to instruction-tuning, where we continue to train language models, using diverse training data synthesized from real tables, in the form of (`instruction, table, completion`), as demonstrations for language models to learn how to correctly perform table-tasks. This process is illustrated on the right of Figure 3.

Through extensive experiments, we show that "table-tuning" is promising as our resulting Table-GPT models are:

(1) <u>*Strong table models*</u>: Table-GPT substantially outperforms 175B GPT-3.5 (text-davinci-002) and ChatGPT (text-chat-davinci-002), on a wide range of seen and unseen table-tasks (data transformation, data cleaning, data profiling, data imputation, table-QA, etc.), as we summarize in Table 2 and Figure 8/Figure 9;

(2) <u>*Generalizable to new tasks*</u>: Table-GPT can respond well to novel and unseen table-tasks, similar to how Chat-GPT can generalize and respond to new and unseen NLP-tasks, like illustrated in Figure 5. We find that even on NLP benchmarks like GLUE, which are not our focus (not table tasks) and completely out-of-domain during fine-tuning, we observe Table-GPT to show strong improvements over GPT-3.5, underscoring the good generalizability of our approach.

We perform an extensive number of train and test experiments (with many failed attempts), and we report the lessons learned in the process. We believe Table-GPT are just first steps in the new table-tuning direction, and hope our effort can serve as a springboard for new research in the area.

Contributions. We make the following contributions in this work:

- We propose a new "table-tuning" paradigm, specifically designed to enhance language models' ability to perform table-tasks, using diverse table-tasks synthesized from real tables.
- We develop task-level, table-level, instruction-level, and completion-level data augmentation techniques for table-tuning, which we show are crucial to avoid over-fitting and ensure the generalizability of Table-GPT.
- We show that Table-GPT not only excels on table-tasks in both zero-shot and few-shot settings out of the box, but can also serve as a "table foundation model" and used as a better starting point than vanilla GPT, for down-stream single-task optimizations such as task-specific fine-tuning and prompt-engineering.
- We release our code, training data, as well as an extensive evaluation benchmark for table-tasks to facilitate future research [6].

2 PRELIMINARIES

We will start with a review of language models, and the use of language models for table-tasks.

2.1 Language models

There are two popular styles of language models today, known as the decoder and encoder-style, both derived from the original transformer architecture [70].

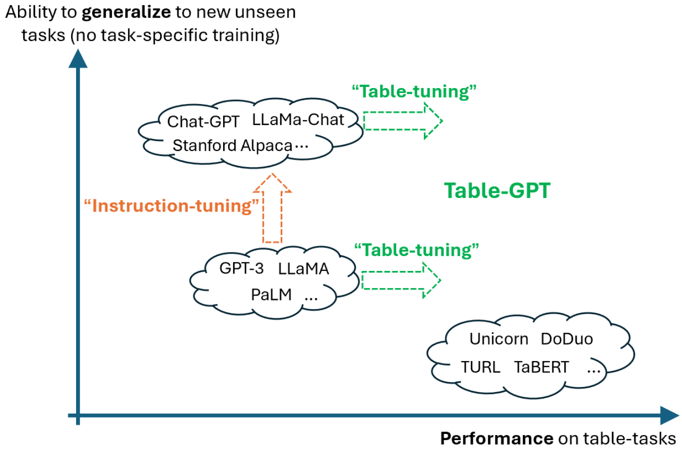


Fig. 4. Visual comparison of Instruction-tuning vs. Table-tuning. Instruction-tuning improves model "generalizability", to follow diverse human-instructions and perform unseen tasks (y-axis). Our proposed table-tuning is similar in spirit but aims to improve model ability to understand tables and perform table-tasks (x-axis).

Encoder-style language models. One class of popular language models, including the well-known BERT [22] and RoBERTa [46], use only encoders from the transformer, and are pre-trained on large amounts of texts to effectively represent the semantics of texts using embedding vectors.

Down-stream tasks: Task-specific fine-tuning. To use encoder-style models like BERT for down-stream tasks, *task-specific fine-tuning* is generally employed [26, 45], which continues to fine-tune (or train) BERT-like models for a given task, using task-specific labeled data. For example, suppose the downstream task is sentiment analysis of Yelp restaurant reviews, then labels in the form of ("`The food is amazing`", "`positive`"), ("`The service is slow`", "`negative`"), are needed to fine-tune BERT-like models for the desired outcome [22, 60].

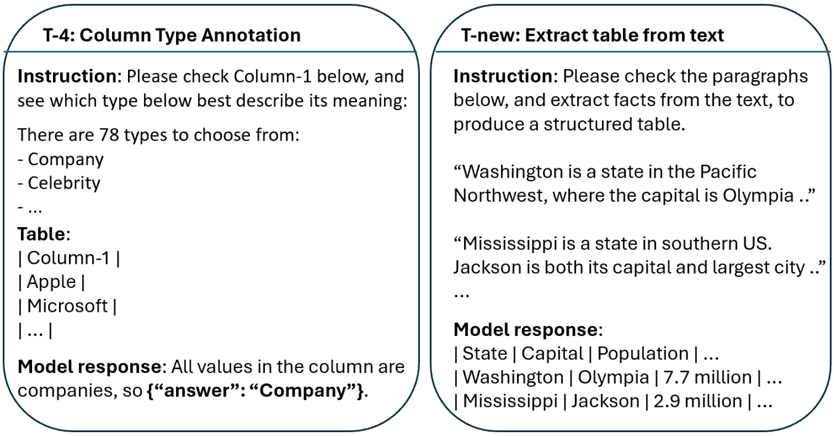
Crucially, when the target input data or the desired output changes, the labeling effort often needs to repeat for the best performance. For example, if the input data for sentiment analysis changes to IMDB reviews, or if the output needs to include a classification of "`cuisine-type`" for restaurant reviews. While encoder-style language-models are strong models, the need to fine-tune with task-specific labelled data limits its ability to generalize to new unseen tasks [22, 27, 46, 60].

Decoder-style "generative" language models. Another class of decoder-only language models, such as GPT [11] and LLaMa [67], are generative in nature, and are shown to excel in generalizing to new downstream tasks *without* task-specific fine-tuning [11].

Generalize to new tasks: zero-shot and few-shot learning. It was shown in the NLP literature that the decoder-style models (e.g., GPT and LLaMa), especially after instruction-tuning [43, 53, 61, 75–78, 94] (e.g., ChatGPT/InstructGPT [3, 53] and Stanford Alpaca [5]), can adapt to new tasks easily, using just natural-language instructions (e.g., "`classify the sentiments in the following reviews`"), and optionally a few examples. Such an approach can adapt to new datasets (e.g., IMDB vs. Yelp reviews) and new tasks (sentiment-analysis vs. machine-translation), without fine-tuning on labeled data for each specific task, making the decoder-style models more versatile. Figure 4 shows the benefit of "instruction-tuning" in model generalizability, depicted on the y-axis.

2.2 Language models for table tasks

Pioneering work in the database literature have employed language models in various ways to perform table-related tasks.



T-4: Column Type Annotation

Instruction: Please check Column-1 below, and see which type below best describe its meaning:

There are 78 types to choose from:
- Company
- Celebrity
- ...

Table:
| Column-1 |
| Apple |
| Microsoft |
| ... |

Model response: All values in the column are companies, so **{“answer”: “Company”}**.

T-new: Extract table from text

Instruction: Please check the paragraphs below, and extract facts from the text, to produce a structured table.

“Washington is a state in the Pacific Northwest, where the capital is Olympia ..”

“Mississippi is a state in southern US. Jackson is both its capital and largest city ..”
...

Model response:
| State | Capital | Population | ...
| Washington | Olympia | 7.7 million | ...
| Mississippi | Jackson | 2.9 million | ...

Fig. 5. Table-models should ideally “generalize” to new datasets and new tasks. (Left) Column type annotation (CTA): while CTA is a common table-task, the list of target-types to choose from can vary from dataset to dataset (e.g., 78 types in [34], and 107 in [21]). Making table-models to “generalize” to new CTA datasets without needing to retrain is useful. (Right) Extract table from text: a general table-model should act like ChatGPT, in following instructions to perform ad-hoc unseen table-tasks like this. Our goal in building Table-GPT is to be generalizable to both new datasetes and new tasks.

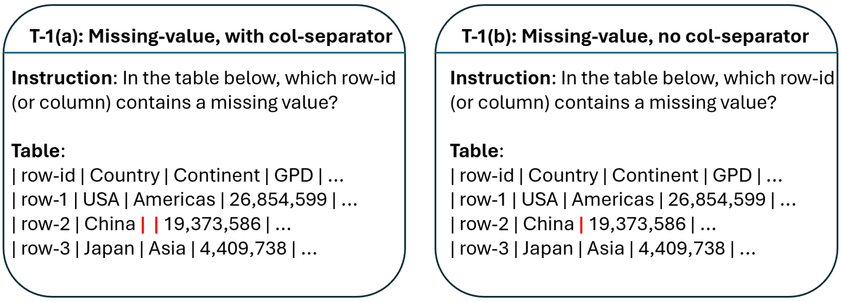
Encoder-style language models for table tasks. There is a long and fruitful line of research (e.g., TURL [21], TaBERT [86], Ditto [44] and Doduo [64]), where table-models are trained based on encoder-style BERT-like models, which are shown to perform well on various *classification-oriented table-tasks*. Such models, however, are unable to perform *generative table-tasks* such as NL-2-SQL or table-QA, given the encoder-style nature of their base models.

Furthermore, similar to their BERT-like base models, in order to generalize to a new dataset or a new task, these encoder-style models typically need to be fine-tuned with task-specific labeled data for the best performance. As a concrete example, for the table-task of “column-type-annotation” [21, 64], in order to move from one dataset with 78 semantic types [34], to another dataset with 107 semantic types [21], new labeled data have to be obtained, so that the models can be fine-tuned to generate the correct output with 107 classes [21]. In contrast, a key goal we aim to achieve, is to build table-models that can adapt to new datasets and tasks *without* task-specific fine-tuning, using only high-level instructions (like how humans interact with ChatGPT), as illustrated in Figure 5.

Decoder-style language models for table tasks. With the success of decoder-style language models such as GPT-3 and ChatGPT, which are shown to perform tasks out-of-the-box with instructions only, pioneering research in the database field develop “*prompt-engineering*” techniques for table-tasks (e.g., [38, 52, 56]), which carefully select instructions and examples in the prompt, to improve the performance of vanilla language models on table-related tasks. Fine-tuning for select table tasks has also been proposed (e.g., [83, 89]).

Table-tuning for table-tasks. In contrast to prompt-engineering that optimizes prompts, our proposed “table-tuning” explores an orthogonal direction, where we continue to train the underlying language models, *for once only* (not task-specific), so that the resulting model perform better on a range of table-tasks. This is complementary to prompt-engineering, because carefully-engineered instructions and examples can continue to benefit both the vanilla GPT as well as our Table-GPT, as we will show in our experiments.

Figure 4 shows a visual comparison of table-tuning vs. instruction-tuning. Whereas instruction-tuning improves model generalizability to follow human instructions (y-axis), table-tuning improves



T-1(a): Missing-value, with col-separator

Instruction: In the table below, which row-id (or column) contains a missing value?

Table:
| row-id | Country | Continent | GPD | ...
| row-1 | USA | Americas | 26,854,599 | ...
| row-2 | China | | 19,373,586 | ...
| row-3 | Japan | Asia | 4,409,738 | ...

T-1(b): Missing-value, no col-separator

Instruction: In the table below, which row-id (or column) contains a missing value?

Table:
| row-id | Country | Continent | GPD | ...
| row-1 | USA | Americas | 26,854,599 | ...
| row-2 | China | 19,373,586 | ...
| row-3 | Japan | Asia | 4,409,738 | ...

Fig. 6. Two variants of (T-1) Missing-cell-identification. (Left) T-1(a): We remove a random cell from a table, but keep its column-separator. The presence of "| |" indicates a missing cell, which should be easy to identify. (Right) T-1(b): We remove a random cell, as well as its column-separator, which is a common but challenging issue in CSV parsing [23, 69, 71].

language models ability to understand tables and perform table-tasks (x-axis). Crucially, as we will show, our table-tuned models remain to be general and capable of following human-instructions to perform ad-hoc table-tasks (e.g., in Figure 5), similar to how ChatGPT would behave. In other words, with Table-GPT we aim to get the "best of both worlds", with both good table-task performance, and generalizability to ad-hoc new tasks.

3 CAN LANGUAGE MODELS "READ" TABLES?

Since language models like GPT are pre-trained predominantly on natural language text, we start by asking whether language models can read tables reliably, which are different from text in many ways.

One-dimensional (text) vs. two-dimensional (tables). Language models are trained mostly on natural language text (e.g, books and web pages) and programming code (e.g., GitHub), both of which that are *one-directional* that is meant to be read *left-to-right*, toke-by-token, in a sequential manner.

In contrast, relational tables are *two-dimensional* with rows and columns, where reading *top-to-bottom* vertically, to see column-headers and other values in the same column (which may be far away in the context window), is crucial for many table-tasks.

Consider the task of Data-Imputation [10, 50] (T-8 in Table 2), which is to infer a missing value in a table cell, like shown in the example of Figure 2 (Right). At least for humans, it is natural to look vertically in the horizontal direction, to see the column-header ("`continent`" in this case), as well as other values in the same column (e.g., "`Americas`"), before one can make a guess for the missing value.

Even for tasks like Table Question-Answering [55, 66] (T-3 in Table 2), which is traditionally an NLP problem, examples like Figure 2 (Left) shows that reading vertically in a column (e.g., for values in the "`art`" column) is similarly important.

To test language models' ability to read tables in the columnar direction, we design a few simple tests. In the first test, referred to as "Missing-value-identification" (T-1 in Table 2), we sample a real table T with no missing cells, and remove a random cell from T. We then produce two variants of the test, like shown in Figure 6:

T-1(a): we keep the column separator of the missing cell and ask language-models to identify the row-id/column-header of the missing cell, like shown in Figure 6 (Left), which seems simple;

T-1(b): We remove the column separator of the missing cell also, and ask the same question, like in Figure 6 (Right). This is a common situation in CSV parsing that can be challenging [23, 69, 71], as one needs to align values vertically to see which column is missing a value. (In this case, humans can see that the countries "`USA`" and "`China`" should align, the GPD numbers should align, so there must be a missing cell in "`row-2`", in between "`China`" and "`19,373,586`", for the column "`Continent`").

We repeat these two tests 1000 times, using 1000 randomly sampled real tables. Table 1 shows the result of this test. We can see that it is clearly challenging for language models to read tables in the column direction, where the accuracy with and without column-separator is 0.38 and 0.26, respectively. Even with column-separators and explicit few-shot demonstrations, the model is only able to get half of the tests right (0.51).

Table 1. Accuracy of vanilla GPT-3.5 (`Text-Davinci-002`), on the task (T-1) Missing-value-identification shown in Figure 6.

(T-1): Missing Value Identification	Find col-header tests		Find row-id tests	
	no col-sep	with col-sep	no col-sep	with col-sep
GPT-3.5 (zero-shot)	0.26	0.30	0.76	0.87
GPT-3.5 (few-shot)	0.38	0.51	0.77	0.91

In the row-direction, the model's ability to identify a missing cell is clearly better, though still not great, especially in the "no col-separator" setting.

To ensure that the language models are not confused by what we mean in "missing cell", we create a second, even simpler test, called Column-Finding (T-2 in Table 2), illustrated by the example in Figure 1 (Right), where we ask the model to find the column-header of a specific value, which appears exactly once in the table, for 1000 randomly sampled tables. Our result show that the accuracy of GPT-3 is similarly low (0.46), confirming that language models' ability to read two dimensional tables is likely insufficient.

Order-sensitive (text) vs. permutation-invariant (tables). In addition, we observe that natural-language texts tend to be *order-sensitive*, where swapping two tokens will generally lead to different meanings. In comparison, tables tend to be *permutation-invariant*, where swapping two rows or two columns, should generally not change the semantic meaning of the table.

As a result, we find that when applying language-models to table-tasks like Entity-Matching and Error-Detection, the predictions can be sensitive to the order in which columns are presented in the input tables, even if we only slightly re-order the columns. Because the decisions for tasks like Entity-Matching and Error-Detection should not depend on the order of columns, we believe it also points to sub-optimal behaviour of language models on tables.

Other differences. There are a number of additional aspects that make tables different from text. For example, table-cells tend to be short-form entity-names or phrases, which when serialized in a row, will typically be different from natural-language sentences found in text documents. Furthermore, because values in the same column tend to have homogeneous values, pairs of columns encode regular semantic relationships, which is another property not common in texts. All of these differences motivate us to optimize language models with table-tuning.

4 TABLE-TUNING FOR TABLE-GPT

We propose a new table-tuning paradigm to enhance language models' ability to understand tables.

4.1 Overall approach: Synthesis-then-Augment

Like discussed earlier, our table-tuning is inspired by the success of "*instruction-tuning*" from the NLP literature [53, 76, 78], illustrated in Figure 3 (Left), where diverse training data in the form of "`(instruction, completion)`" pairs are used as train data.

Table 2. **A summary of 18 table-related tasks, which we compile and synthesize for Table-GPT**. Note that T1-T4 are held-out "test-only tasks" used exclusively for testing (unseen during training), to test model generalizability to completely new tasks. **[Task categories]**: These tasks cover diverse areas such as: table understanding, table-QA, table matching, table cleaning, table transformation, etc. **[Table Data]**: we choose to "synthesize" table tasks from diverse real tables when possible (e.g., when ground-truth can be produced automatically), so that we can hold existing labeled benchmarks completely separate for testing only (e.g., the TDE benchmark is used for testing only). When ground-truth cannot be automatically produced (e.g., entity-matching, table-QA, NL-to-SQL, etc.), we use labeled data as training, but still hold test benchmarks completely separate (e.g., the DeepM benchmark is used exclusively for testing not for training).

Task-name	Task description (related work)	Task category	Table data	Train/Test
T-1: Missing-value identification (MV)	Identify the row and column position of the only missing cell in a given table	Table understanding	Synthesized	Test only tasks
T-2: Column-finding (CF)	Identify the column-name of a specific value that appears only once in a given table	Table understanding	Synthesized	
T-3: Table-QA (TQA)	Answer a natural-language question based on the content of a table ([14, 55, 66])	Table QA	WikiTQ [55], SQA [35]	
T-4: Column type annotation (CTA)	Find the semantic type of a column, from a given list of choices ([21, 34, 85])	Table understanding	Sherlock [34], TURL [21]	
T-5: Row-to-row transformation (R2R)	Transform table data based on input/output examples ([29, 30, 36])	Data transformation	Synthesized, TDE [30]	Train & Test tasks
T-6: Entity matching (EM)	Match rows from two tables that refer to the same real-world entity ([44, 51, 54, 92])	Table matching	Magellan [20], DeepM [1]	
T-7: Schema matching (SM)	Match columns from two tables that refer to the same meaning ([39, 48, 57])	Table matching	Synthesized, DeepM [39]	
T-8: Data imputation (DI)	Predict the missing values in a cell based on the table context ([10, 50])	Data cleaning	Synthesized	
T-9: Error detection (ED)	Detect data values in a table that is a likely error from misspelling ([17, 58])	Data cleaning	Synthesized, new labeled benchmark [6]	
T-10: List extraction (LE)	Extract a structured table, from a list that lacks explicit column delimiters [12, 16, 24]	Data transformation	Synthesized	Train only tasks
T-11: Header value matching (HVM)	Match column-headers with its data values drawn from the same table	Table matching	Synthesized	
T-12: Natural-language to SQL (NS)	Translate a natural-language question on a table into a SQL query ([82, 87])	NL-to-SQL	WikiSQL [93]	
T-13: Table summarization (TS)	Produce a natural-language summary for the content in a table	Data augmentation	Synthesized	
T-14: Column augmentation (CA)	Augment a table with additional columns compatible with a given table	Data augmentation	synthesized	
T-15: Row augmentation (RA)	Augment a table with additional rows compatible with a given table	Data augmentation	synthesized	
T-16: Row/column swapping (RCSW)	Manipulate a given table, by swapping the position of two rows or columns	Table manipulation	Synthesized	
T-17: Row/column filtering (RCF)	Manipulate a given table, by filtering on given rows or columns	Table manipulation	Synthesized	
T-18: Row/column sorting (RCS)	Manipulate a given table, by performing sorting on given rows or columns	Table manipulation	Synthesized	

Our proposed *table-tuning*, as illustrated in Figure 3 (Right), is similar in spirit, but we aim to improve language-models' ability on tables using diverse "`(instruction, table, completion)`" triples, where each such triple defines an instance of a *table-task*:

Definition 1. An instance of a *table-task*, denoted by t, is defined as a triple $t = (Ins, T, C)$, where Ins is the natural-language instruction that describes the table-task, T is the input table on which the task is to be performed, and C is the expected completion from performing the instructed task on the table T.

Example 1. The examples in Figure 1, Figure 2, and Figure 3, show simple examples of table-tasks, defined by the (Ins, T, C) triples, which correspond to (`instruction, table, completion`) shown on the figures, respectively. Note that the completion C can be natural-language texts (with embedded to assist answer-parsing), tables, or a combination of both.

Algorithm 1: Synthesize table-tasks for table-tuning

```
   input  : A corpus of diverse real tables C, a set of table-task types S
   output : Diverse synthesized table-tasks $A = \{(Ins, T, C)\}$
1  $D \leftarrow \{\}, A \leftarrow \{\}$
2  foreach $T \in \mathbf{C}, S \in \mathbf{S}$ do
3      $(Ins, T, C) \leftarrow$ Synthesize-Table-Task$(S, T)$ // (Section 4.2)
4      $D \leftarrow D \cup (Ins, T, C)$
5  foreach $(Ins, T, C) \in D$ do
6      $Ins' \leftarrow$ Augment-Instruction$(Ins)$ // (Section 4.3)
7      $T' \leftarrow$ Augment-Table$(T)$ // (Section 4.3)
8      $C' \leftarrow$ Augment-Completion$(C)$ // (Section 4.3)
9      $A \leftarrow A \cup (Ins', T', C')$
10 return $A$
```

The challenge, however, is that prior work on instruction-tuning have shown that the quality and diversity of the training "`(instruction, completion)`" pairs is crucial [53, 67], to the extent that companies hired armies of human labelers to manually generate high-quality completions, (e.g., instruction: "`write a bed-time story with a bear goes to beach`", completion: `an-actual-story-with-bears`) [53].

We would like to replicate the success of instruction-tuning in the table domain, but ideally without the expensive human labeling.

Reusing existing benchmark data: insufficient diversity. One approach to generate table-tasks, is to use existing benchmark data published in the database literature (similar efforts were made in the NLP literature for instruction-tuning [78]).

However, we found that when used as *training-data* for language-models, the existing benchmark data have:

(1) *limited task-diversity*: as the literature tends to focus on a few selected table-tasks that are hard (e.g., entity-matching, data-transformation, etc.); and

(2) *limited data-diversity*: as benchmark data are typically labeled manually by researchers, only on a few datasets, which is sufficient as a benchmark for evaluations, but insufficient when we want to use them as "training data" for language models.

Our attempt to use only existing benchmark data for table-tuning leads to over-fitting, due to the lack of task and data diversity.

Our approach: Synthesis-then-Augment. We therefore propose a "*synthesize-then-augment*" approach to create diverse table-tasks using real tables as listed in Table 2, which can be used as training data to demonstrate the desirable behavior on tables for language models.

The main steps of our synthesize-then-augment approach is shown in Algorithm 1. First, we sample a table $T \in \mathbf{C}$ from a large corpus of real tables $\mathbf{C}$, and a type of table-task $S \in \mathbf{S}$. From the (T, S) pair, we synthesize an instance of a table-task $t = (Ins, T, C)$ (Line 3), which is the task-synthesis step we will describe in detail in Section 4.2. From the set of diverse instances of table-tasks created (Ins, T, C), we then proceed to "augment" the tasks, at instruction/table/completion levels (Line 6-8), which is the step that we will describe in Section 4.3. The resulting table-tasks $A = \{(Ins', T', C')\}$ become the training data we use for table-tuning.

4.2 Synthesize diverse table-tasks

We now describe how we synthesize diverse instances of table-tasks $t = (Ins, T, C)$ (Line 3 of Algorithm 1) using real tables.

We develop two complementary methods that (1) synthesize new types of table-tasks for *task-diversity*, and (2) synthesize new table test-cases of existing table-tasks for *data-diversity*. We will discuss each in turn below. Details of the synthesized tasks and examples can be found in our technical report [7].

Synthesize new types of table-tasks (for task-diversity). Since our goal is to exercise language-models ability to understand two-dimensional tables, we believe it is not necessary to focus exclusively on challenging table-tasks that have been the focus of the literature [58]. Instead, we propose a number of table-understanding, augmentation, and manipulation tasks that are easy to synthesize, leveraging large amounts of real tables that already exist. Specifically, we crawled 2.9M high-quality web-tables (e.g., Wikipedia), referred to as $\mathbf{C}^{wt}$, and 188K database-tables (extracted from BI data models), referred to as $\mathbf{C}^{db}$, and we synthesize table-tasks based on real tables sampled from these corpus.

We will go over the list of synthesized table-tasks below:

(T-13) Table summarization (TS). Since web-tables often have descriptive titles, we synthesize a task to summarize the content of a table. Specifically, we sample $T \in \mathbf{C}^{wt}$ whose table-title $title(T)$ is neither too long nor too short, from which we create the table-summarization task $TS(T)$ as:

$$TS(T) = (Ins^{TS}, T, title(T))$$

where Ins^{TS} is the canonical human-written instruction to describe the TS task (e.g., "`Please provide a succinct summary for the table below`", which will be further augmented in Section 4.3), T is a real table, and $title(T)$ is the expected completion for the task.

This task is designed to use real human-written titles for tables, to enhance models ability to read and understand table. Note that although we use $title(T)$ as the expected completion, with enough data/task diversity, it only "nudges" language-models in the right direction, but does not over-constrain language-models to over-fit on such completions, which is similar to how training data in the form of ("`write a bed-time story with a bear`" ⟶ `an-actual-human-written-story`) does not over-constrain/over-fit models in instruction-tuning too.

(T-14) Column augmentation (CA). Augmenting a table with additional plausible columns is another task that can exercise models' ability on tables, while being simple to synthesize since we have lots of real tables in $\mathbf{C}^{wt}$ and $\mathbf{C}^{db}$. Specifically, we take the first k columns in a table T, denoted as $C_{[1,k]}(T)$, and ask the language-models to generate the $(k+1)$-th column $C_{k+1}(T)$, which can be written as:

$$CA(T, k) = (Ins^{CA}, C_{[1,k]}(T), C_{k+1}(T))$$

where Ins^{CA} is the natural-language instruction for the CA task. This task exercises a model's ability to generate realistic columns that are semantically compatible with an existing table T.

(T-15) Row augmentation (RA). Similar to Column-augmentation, we synthesize a Row-Augmentation task where we sample a table T, and ask the model to generate the $(k+1)$-th row, given the first k rows, which is written as:

$$RA(T, k) = (Ins^{RA}, R_{[1,k]}(T), R_{k+1}(T))$$

This task exercises a model's ability to synthesize realistic rows compatible with an existing table T, which need to align vertically with existing values in the table.

(T-16) Row/column swapping (RS/CS). In this task, we ask the models to perform a table-manipulation step, where given a sampled table T, we provide an instruction to swap the i-th and j-th row. We programmatically generate the resulting table from the swap operation, denoted as $Swap(T, R_i, R_j)$, which is the target "`completion`". The Row-swapping task $RS_{i,j}(T)$ is written as:

$$RS_{i,j}(T) = (Ins^{RS}, T, Swap(T, R_i, R_j))$$

We similarly synthesize the Column-swapping task $CS_{i,j}(T)$ as:

$$CS_{i,j}(T) = (Ins^{CS}, T, Swap(T, C_i, C_j))$$

We note that tasks like Row/Column-swapping may seem simple, when they can be performed by humans either programmatically or manually through an UI (e.g., using menu options in spreadsheet software). However, language models frequently struggle on such tasks, and we are only intending to use these manipulation tasks as "training data" for language models to better understand tables, because in the end, regardless of whether we want language-models to generate code or text, input tables still need to be serialized as text and consumed by language models, where the ability to read tables correctly is important.

(T-17) Row/column filtering. In this table-manipulation task, we ask models to filter down to specific rows/columns on a table T, based on given row indexes I_r (e.g., the second and fifth rows), and column indexes I_c (e.g., the "`country`" and "`population`" columns):

$$RF(T, I_r) = (Ins^{RF}, T, R_{[I_r]}(T))$$

$$CF(T, I_c) = (Ins^{CF}, T, C_{[I_c]}(T))$$

These tests are again meant to exercise models' ability to manipulate tables, in both horizontal and vertical directions.

(T-18) Row/column sorting (RS/CS). In the sorting tasks, we ask models to sort rows in a table T, based on values in a column C, where the expected output table can be programmatically generated as the expected completion, which we write as $Sort_C(T)$:

$$RS_C(T) = (Ins^{RS}, T, Sort_C(T))$$

Similarly, we have a task to sort columns in a table T, based on column-headers H, to produce a column-header sorted table $Sort_H(T)$:

$$CS(T) = (Ins^{CS}, T, Sort_H(T))$$

We note that the sorting tasks are fairly challenging for language-models – while we do not expect models to be perfect on such tasks, they exercises models' ability to manipulate tables nevertheless.

(T-11) Header-value matching (HVM). In this task, we sample a table T, remove all its column headers H to produce the corresponding table without headers, $\overline{T}$. We then shuffle the headers H, and ask models to correctly fill H into T', to get back the original T:

$$\text{HVM}(T) = (Ins^{HVM}, \overline{T}, T)$$

Like other tasks above, we can synthesize HVM automatically, using large numbers of real tables. It is intended to help models understand the correspondence between column-headers and values.

Discussions. We observe in our experiments, that these tasks synthesized from real tables improve the task- and data-diversity, and lead to better model generalizability.

Our list of synthesized table-tasks, however, is clearly not meant to be exhaustive, and we believe it is only a starting point. With some creativity, many more tasks can be synthesized to further improve table-tuning. For comparison, the NLP community has amassed over 1000 tasks for instruction-tuning through community efforts [18], where they show that having more diverse tasks always help instruction-tuning.

Synthesize new table cases of existing tasks (for data-diversity). There are a number of important existing types of tasks, such as data-transformation, entity-matching, etc., that are extensively studied in the database literature. Given their importance, we want to include these tasks in table-tuning, in the same "`(instruction, table, completion)`" format. However, like mentioned earlier, existing benchmarks for these tasks are typically manually labeled on only a few datasets, which are meant for evaluations, but too limited as "training data" for language models. (The other problem is if we use existing benchmarks as training data, then there is no easy way to evaluate the resulting models on these tasks).

We therefore synthesize new table test-cases for these existing task types, using real tables sampled from $\mathbf{C}^{wt}$ and $\mathbf{C}^{db}$.

(T-5) Row-to-row Data Transformation (R2R) [29, 30]. To synthesize diverse tables involving transformations, we run a production-quality program-synthesizer, on web-tables sampled from $\mathbf{C}^{wb}$, to identify tables $T \in \mathbf{C}^{wb}$ where there exist two disjoint groups of columns, $C_{in} \subset T$ and $C_{out} \subset T$, such that using a program inferred from program-synthesis P, we have $P(C_{in}) = C_{out}$ for all rows in T (e.g., `(first-name, last-name)` → `(full-name)` in the same table). For such a table T, we remove a random value $v \in C_{out}$, to produce T_{-v} where v is missing. We then synthesize a task $R2R(T)$ as:

$$R2R(T) = (Ins^{R2R}, T_{-v}, T)$$

where given T_{-v} as the input, we want to the model to infer the transformation and fill in the missing v to get back the original T.

(T-7) Schema Matching (SM) [57]. To synthesize new table test cases for schema matching, we sample a real table T, where we take the first k rows from T to produce $T_1 = R_{[1,\ k]}(T)$, and then the next k rows from T to produce $T_2 = R_{[k+1,\ 2k]}(T)$. We then "paraphrase" the column-headers in T_2 using GPT (e.g., "`company names`" → "`enterprises`", "`emp-id`" → "`employee identifier`", etc.), where we use M to denote the mapping of the paraphrased column-headers between (T_1, T_2). Finally, we shuffle the columns in T_1 and T_2 to create a task $SM(T)$ as:

$$SM(T) = (Ins^{SM}, (T_1, T_2), M)$$

Like other tasks, this type of mapping tasks can also be synthesized on diverse real tables, and used as training data for table-tuning.

(T-8) Data Imputation (DI) [10, 50]. For data imputation, we randomly sample a real table T, and then remove a random value $v \in T$, to produce T_{-v}. The task $DI(T)$ is then to predict the missing v from the remaining table context in T_{-v}:

$$DI(T) = (Ins^{DI}, T_{-v},\ v)$$

While not all missing values v in synthesized DI tasks can be reliably predicted, it nevertheless trains models to leverage two-dimensional table context for predictions (this is analogous to how next-tokens cannot always be reliably predicted from texts, yet next-token prediction is still a good training-objective for language modeling).

(T-9) Error Detection (ED) [58]. To synthesize error-detection tasks, we sample a real table $T \in \mathbf{C}^{wt} \cup \mathbf{C}^{db}$, and with probability p, we inject a misspelling error, by generating a modified $\tilde{T}$ where we replace a sampled value $v \in T$ with its misspelled version v' (using an existing package [4]). With the remaining probability $1 - p$, we keep T as is without injecting misspellings. The task $ED(T)$ is then:

$$ED_p(T) = \begin{cases} (Ins^{ED}, \tilde{T},\ \{v'\}), & \text{with probability } p \\ (Ins^{ED}, T,\ \emptyset), & \text{with probability } 1 - p \end{cases}$$

the goal of this task is not only to identify the misspelled $v' \in \tilde{T}$ based on the table context, when an error exists, but also learn to not over-trigger and produce false-positives (a common problem with vanilla models), when no errors are present.

(T-10) List extraction (LE) [16, 24]. For the task of extracting tables from list data that lack explicit column-delimiters, we sample a table T, and replace all column separators with white spaces, to produce T's unsegmented list-form $L(T)$. The task $LE(T)$ is then:

$$LE(T) = (Ins^{LE}, L(T), T)$$

which is to generate the ground-truth segmentation and get back the table T, from the unsegmented $L(T)$. Being able to align values in the vertical direction in a table, is crucial to perform this task.

4.3 Augmentation of synthesized table-tasks

From diverse synthesized table-tasks (Ins, T, C), we then perform task-augmentations at different levels, corresponding to steps in Line 6-Line 8 of Algorithm 1, where the goal is to create even more data diversity. We will go over these augmentations in turn below.

Instruction-level augmentations. At the instruction level, because it was shown that using the same instruction repeatedly across training-data instances can lead to over-fitting [76–78], we augment the canonical human-written instruction for each task-type using generative models like GPT. Specifically, we ask GPT to paraphrases the canonical instruction into many alternative instructions to describe the same task.

For example, for the task-type (T-13): Table-Summarization (Section 4.2), the canonical human-written instruction is: "`Please look at the table below and provide a title that can summarize the table`". We generate many alternative instructions from language-models, such as "`Please examine the table below and give it a descriptive title`", similar to how instructions are augmented in instruction-tuning [76]. We then sample variants of the instructions for the same task-type, to increase instruction diversity (Line 6).

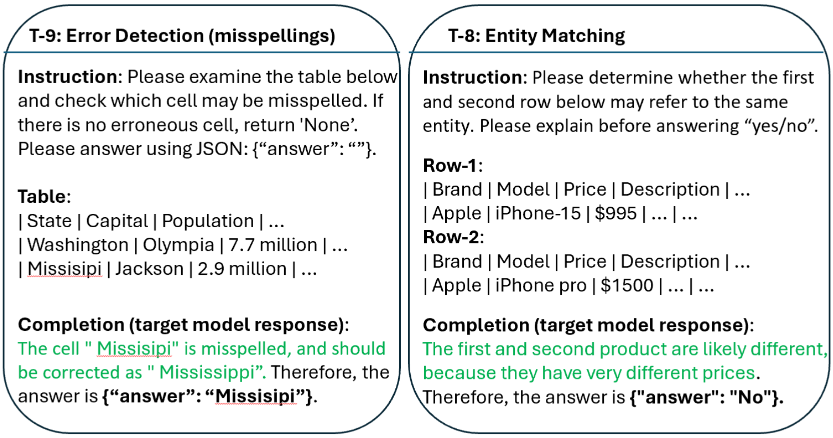
Table-level augmentations. At the table-level, we know that tables should generally be "invariant" to rows and columns permutations (Section 3), so at the table-level we can perform augmentation operations such as row and column permutations, which should usually not change the semantics of the table.

When we populate the training data with an original table-task, $t = (Ins, T, C)$, as well as its augmented version $t' = (Ins, T', C)$, where T' is a permuted version of T (which has the same semantic meaning of T, and therefore the same completion C), the hope is that language-models can learn to be less sensitive to permutations (the orders of tokens which may be important for texts, but the orders of columns are much less so for tables). This should lead to more stable behaviors and more optimized performance on table-tasks.

Completion-level augmentations. At the completion-level, we observe that for more complex table-tasks (e.g., entity-matching and error-detection), performing reasoning-steps (analogous to chain-of-thought [74, 79]) can lead to better task performance. Therefore, for a synthesized training task (Ins, T, C), we augment the completion C by inserting reasoning-steps before the final answer, like illustrated in Figure 7, for models to learn to perform reasoning on complex table tasks. Here we augment by leveraging ground-truth and language-model-generation, as follows.

Completion-augmentations by ground-truth. We observe that for the task of (T-9) Error-Detection, vanilla language models are prone to produce false-positives, where the models would confidently predict abbreviations or uncommon names as misspelled when no misspellings exist. Our intuition is that if we require models not only to predict a misspelled value v', but also explain the prediction with the corrected version of v', e.g., "`Missisipi`" should be "`Mississippi`" like shown in Figure 7 (left), then models will be forced to produce factual predictions grounded on actual corrections, leading to higher accuracy (because it is hard to generate plausible corrections for false-positive detection).

Therefore, in synthesized training tasks, we use the original value v before we inject typos from the synthesis-step (Section 4.2) as the target correction, to insert a reasoning step to the effect of: v' is misspelled and the corrected value should be v, before the predicted answers in the



T-9: Error Detection (misspellings)

Instruction: Please examine the table below and check which cell may be misspelled. If there is no erroneous cell, return 'None'. Please answer using JSON: {"answer": ""}.

Table:
| State | Capital | Population | ...
| Washington | Olympia | 7.7 million | ...
| Missisipi | Jackson | 2.9 million | ...

Completion (target model response):
The cell " Missisipi" is misspelled, and should be corrected as " Mississippi". Therefore, the answer is **{"answer": "Missisipi"}**.

T-8: Entity Matching

Instruction: Please determine whether the first and second row below may refer to the same entity. Please explain before answering "yes/no".

Row-1:
| Brand | Model | Price | Description | ...
| Apple | iPhone-15 | $995 | ... | ...
Row-2:
| Brand | Model | Price | Description | ...
| Apple | iPhone pro | $1500 | ... | ...

Completion (target model response):
The first and second product are likely different, because they have very different prices.
Therefore, the answer is **{"answer": "No"}**.

Fig. 7. Augmented completions: for complex tasks like (T-9) Error-detection and (T-6) Entity-matching, we insert reasoning-steps (marked in green) before answers, in the completion of our synthesized training tasks.

completion, like shown in Figure 7 (left). Such augmented-completions, when used in table-tuning, encourage language-models to reason on complex table-tasks, and reduce false-positives on tasks like error-detection.

Completion-augmentations by language-model generation. For tasks like (T-6) Entity-matching, we find that when prompting language-models to reason "step-by-step" and explain before producing yes/no answers, like shown in the completion of Figure 7 (right), it generally improves the quality of the result, similar to what was observed in the NLP literature (e.g., chain-of-thought reasoning [79]). However, vanilla language-models can still frequently generate incorrect answers or reasoning-steps for a given pair of rows, making it unsuitable to use language-model generation directly to augment our completions.

On training EM datasets (held separate from testing datasets), we instead give language models the actual ground-truth decision (match/non-match) for each pair of input row, for it to generate a chain of reasoning that is more likely to be correct (since the ground-truth answer is already known). We insert the generated reasoning step (shown as the green paragraph in Figure 7 (right)) before the original completion C, which when used to train language-models, can encourage them to learn to reason with the right steps on complex table tasks like entity-matching.

Additional augmentations. We perform additional types of augmentations, including "*template-level augmentation*", where we mix zero-shot task template and few-shot task template (which appends multiple input-table/output-completion examples after the instruction Ins), as well as "*task-level augmentation*" by synthesizing new types of table-tasks like discussed above, all of which aim to improve task/data diversity in the training data for table-tuning.

4.4 Table-GPT as "table foundation models"

Using the synthesis-then-augment approach in Algorithm 1 as described in previous sections, we produce diverse table-tasks $A = \{(Ins, T, C)\}$. We can now continue to train language models such as GPT, using serialized (Ins, T) as the "prompt" (we will explore different ways to serialize T in our experiments), and C as the "target completion" that we want language models to learn from (by minimizing language-modeling loss subject to regularization). This continues to change a language-model weights until it "fits" the given table-tasks in our training data. We refer to this process as table-tuning (analogous to instruction-tuning in NLP).

Let M be a decoder-style language model, such as GPT and ChatGPT, let TableTune(M) be the table-tuned version of M. We argue that TableTune(M) could serve as a better "table foundation model" than M, if it performs better than M in the following scenarios:

(1) Out of the box zero-shot: when we use only instructions for M or TableTune(M) to perform table-tasks;

(2) Out of the box (random) few-shot: when we use instructions and *randomly selected* few-shot examples to perform table-tasks;

(3) Task-specific prompt-tuning: when we optimize for a downstream task, using a small number of labeled data, by performing prompt-tuning that selects the best instruction and examples;

(4) Task-specific fine-tuning: when we optimize for a downstream task, using a sufficiently large number of labeled data, by performing task-specific fine-tuning on M and TableTune(M).

If table-tuning is effective for language models to better understand and manipulate tables, we expect that TableTune(M) can outperform M on many of the scenarios above, which are the evaluations we want to perform in our experiments below.

Lessons learned. We perform an extensive number of over 1000 train and inference experiments, to table-tune language models, many of which are failed attempts (e.g., resulting models do not generalize well). We report the lessons we learned in a technical report [7] in the interest of space.

5 RELATED WORK

Table-related tasks. There is a fruitful line of research on table-related tasks in the data management literature, addressing a wide array of table tasks, such as schema matching [39, 48, 57], entity matching [20, 41, 44, 51, 54, 92], data transformation [19, 29, 30, 36, 42, 95], data cleaning [17, 31, 33, 49, 58, 59, 63, 73], list extraction [12, 16, 24, 40, 80], column type annotation [21, 34, 65, 85, 88], data imputation [10, 21, 50], table augmentation [84, 90], etc. In the NLP literature, there are additional table-related tasks involving natural languages, such as table-QA [14, 55, 66], NL-2-SQL [82, 87, 93], table summarization [13, 28, 91], among many other important tasks.

In Table-GPT, we aim to produce a general-purpose table model that can generalize to many seen and unseen table tasks, analogous to how ChatGPT can respond to new and unseen user requests. Details of the tasks and datasets used in our study can be found in Section 4 and Section 6.

Language models and Table models. Since the introduction of the transformer [70], a class of encoder-only language models, such as BERT [22] and RoBERTa [46], first emerged as popular choices for NLP tasks. These models are strong in representation but are not generative in nature (unlike GPT-like language models), which typically require "task-specific training/fine-tuning" for each individual downstream task (using task-specific training data).

Table-models built upon "encoder-only" language-models, such as TURL [21] and TaBERT [86], are similarly strong in representation learning, but cannot generalize to new and unseen table-tasks without task-specific training, which is a main limitation that we want to overcome in this work.

Recently, a class of "decoder-only" language models that use only decoder modules from the transformer architecture are gaining popularity, which includes GPT [11], Llama [67], and PaLM [8, 15], among many other models. These models are generative in nature, and are shown to excel in new and unseen natural-language tasks (e.g., using in-context few-shot learning), which obviates the need of task-specific training for each individual downstream task, and is a strong benefit of "decoder-only" language models.

In terms of table-models, while there are existing efforts that leverage GPT-like generative models for table-tasks (e.g., with prompt-engineering [38, 52, 56, 81]), we are not aware of any prior effort that attempts to systematically build *general-purpose* table-models on top of generative GPT-like models, that can generalize to *new and unseen* table tasks (similar to how ChatGPT can respond to new and unseen user requests). Table-GPT is a first attempt in this direction, which we believe is a promising area for future research.

Instruction-tuned language models. ChatGPT and its academic counterparts take the generalizability of GPT-like models one step further, by using an approach known as "*instruction fine-tuning*" (or simply instruction tuning), which fine-tunes language-models using (instruction, expected-completion) pairs, that is shown to greatly enhance the underlying model's ability to follow human instructions and perform unseen tasks. There is a long and fruitful line of research in the NLP literature dedicated to instruction fine-tuning, which include Flan [78], Self-instruct [76], Supernatural-instruction [77], T0 [61], Tulu [75], Self-alignment [43], Instruct-GPT [53], Lima [94], among others. Our "*table-tuning*" approach proposed in this work is inspired by instruction-tuning research from the NLP literature, but tailors to table-related tasks (e.g., exploiting the two-dimensional structure of tables).

Prompt-engineering. In additional to model fine-tuning, "*prompt-engineering*" [9, 81] is an orthogonal class of techniques that improve language-model performance on downstream tasks, by optimizing instructions and examples in the prompt (without changing model weights). In the context of table-related tasks, pioneering prior work [37, 38, 52, 56] have shown that careful prompting can enhance vanilla language models such as GPT-3 on a number of table-tasks.

6 EXPERIMENTS

We perform extensive experiments to evaluate table-tuned GPT relative to vanilla GPT on table tasks. Our source code and training/testing data are released at [6] to facilitate future research.

6.1 Experiment Setup

Models Compared. We test the following models on table tasks.

- *GPT-3.5 (text-davinci-002).* We use the 175B "text-davinci-002" model from OpenAI (released in 2022), as one of the vanilla GPT models that we compare with. Note that our experiments predate the "GPT-3.5-turbo" family of models released in 2023, which are different models.
- *Table-GPT-3.5 (text-davinci-002 +table-tune).* This model is obtained by performing table-tuning on GPT-3.5 (text-davinci-002). We compare the performance of Table-GPT-3.5 with GPT-3.5.
- *ChatGPT (text-chat-davinci-002).* This is a version of the ChatGPT model known as text-chat-davinci-002, available internally for testing. We use this as a second base model for table-tuning.
- *Table-ChatGPT (text-chat-davinci-002 +table-tune).* This is the model we obtain by performing table-tuning on ChatGPT (text-chat-davinci-002), which we compare with the vanilla ChatGPT. We use this second comparison to show the generality of table-tuning to different language models (chat-style vs. completion-style models).

Our overall comparisons consist of two pairs of models that are table-tuned vs. vanilla un-tuned, namely, (GPT-3.5 vs. Table-GPT-3.5) and (ChatGPT vs. Table-ChatGPT).

Hyper-parameters. By default, we use LoRA fine-tuning [32] (with 32 dimensions) and train for 2 epochs. We use a batch size of 32, a learning-rate multiplier of 0.1, and a weight decay of 0.02.

Training tasks and data. By default, we use 14 table-tasks as training, listed as T-5 to T-18 in Table 2. We use the "synthesize-then-augment" approach (Section 4) to generate 1000 synthesized table-tasks per task-type (except T-6: Entity Matching and T-12: NL-to-SQL, where labels are hard to generate, for which we use manually-labeled benchmark data from [20] and [87] as training, but the entire datasets are then never used again in tests). We use a 50:50 mix of zero-shot and few-shot templates.

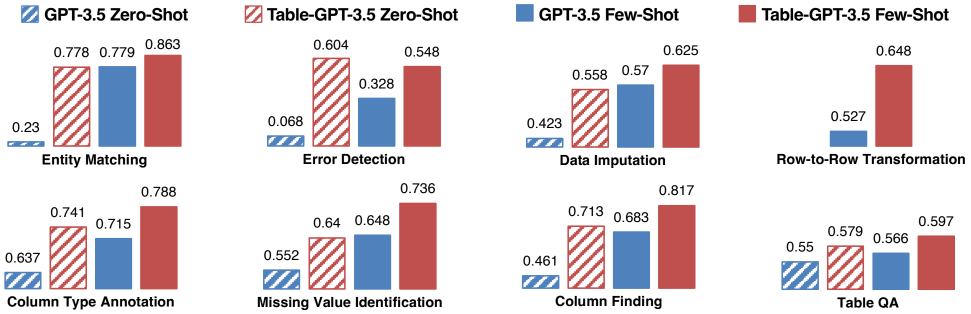


Fig. 8. Quality comparisons between vanilla GPT-3.5 (`text-davinci-002`) and Table-GPT-3.5. All test benchmarks are completely held-out during table-tuning (both the train/test splits of the benchmarks are unseen).

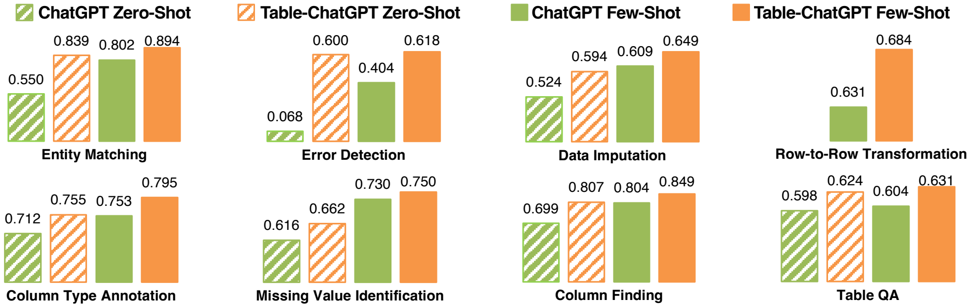


Fig. 9. Quality comparisons between vanilla ChatGPT (`text-chat-davinci-002`) and Table-ChatGPT. All test benchmarks are completely held-out during table-tuning (both the train/test splits of the benchmarks are unseen during table-tuning).

Test tasks and data. We use 4 “unseen tasks” (T-1 to T-4 in Table 2) as our tests. We emphasize that these tasks are *unseen and not included in training* (our training data consists of T5 to T18), so that we can test the generalizability of table-tuned models to new and unseen table tasks. For (T-3) Table QA and (T-4) Column Type Annotation, we use established benchmarks [55] and [21, 34, 68]. For (T-1) Missing Value Identification and (T-2) Column Finding, we use 1000 test cases generated by randomly sampling from a corpus of real spreadsheet tables $\mathbf{C}^{sp}$.

We also evaluate 5 “seen tasks” (T-5 to T-9 in Table 2), which are important tasks that we want table-tuned models to learn from, so as to better understand tables. In order to test such tasks, we use common benchmarks developed by others from the literature, but we ensure that the test benchmarks are held completely separate and unseen in training. For example, for (T-5) Row-to-Row Transformation and (T-7) Schema-Matching, we use synthesized tasks randomly sampled from $\mathbf{C}^{wt}$ and $\mathbf{C}^{db}$ for training, but use manually labeled benchmark data from the literature ([30] and [39]) for testing. For (T-6) Entity-Matching, we use the 784 datasets [20] for training and the DeepMatcher benchmark for testing [1]. For (T-8) Data-Imputation, our training table-tasks are synthesized using tables sampled from $\mathbf{C}^{wt}$ and $\mathbf{C}^{db}$, while tests are generated using a corpus of spreadsheet tables $\mathbf{C}^{sp}$, which are completely different tables (spreadsheets vs. web-tables). For (T-9) Error-Detection, we manually labeled a benchmark with real spreadsheet-tables [6], given the lack of similar benchmarks and its value to Microsoft (again completely held-out in training).

Details. Details of the datasets and the evaluation metrics of these tasks, can be found in the technical report [7]. Our train/test datasets can be downloaded from [6].

Table 3. Detailed quality results of table-tuning, on both GPT-3.5 (`text-davinci-002`) and ChatGPT (`text-chat-davinci-002`). Zero-shot is not applicable to row-to-row transformations, which requires examples (marked as "N.A."). For all "**Unseen**" tasks, the tasks are held-out and unseen during table-tuning. For all "**Seen**" tasks, the task is seen during table-tuning (e.g., using synthesized training data), but the test benchmarks listed in the table below are held completely separate and untouched during training, so we still test model generalizability to new and unseen datasets.

Task Type	**Task**	**Dataset**	**Zero-Shot**		**Few-Shot**		**Zero-Shot**		**Few-Shot**	
			GPT-3.5	+table-tune	GPT-3.5	+table-tune	ChatGPT	+table-tune	ChatGPT	+table-tune
"**Unseen**" (task not seen in training)	CF	Spreadsheets-CF	0.461	**0.713**	0.683	**0.817**	0.699	**0.807**	0.804	**0.849**
	CTA	Efthymiou	0.757	**0.886**	0.784	**0.847**	0.824	**0.882**	0.806	**0.861**
		Limaye	0.683	**0.755**	0.719	**0.853**	0.742	**0.769**	0.832	**0.854**
		Sherlock	0.332	**0.449**	0.528	**0.538**	0.455	**0.483**	0.521	**0.553**
		T2D	0.776	**0.875**	0.830	**0.915**	0.828	**0.887**	0.853	**0.912**
	MV	Spreadsheets-MV-ColNoSep	0.261	**0.294**	0.383	**0.441**	0.299	**0.351**	0.468	**0.474**
		Spreadsheets-MV-ColSep	0.305	**0.457**	0.519	**0.643**	0.422	**0.520**	0.635	**0.665**
		Spreadsheets-MV-RowNoSep	0.768	**0.851**	0.774	**0.882**	0.822	**0.840**	0.859	**0.894**
		Spreadsheets-MV-RowSep	0.875	**0.959**	0.917	**0.976**	0.923	**0.936**	0.960	**0.968**
	TQA	WikiTableQuestion	0.450	**0.486**	0.455	**0.478**	0.513	**0.521**	0.520	**0.528**
		SequentialQA	0.650	**0.672**	0.678	**0.717**	0.683	**0.728**	0.689	**0.733**
"**Seen**" (task seen in training, but test data not seen in training)	DI	Spreadsheets-DI	0.423	**0.558**	0.570	**0.625**	0.524	**0.594**	0.609	**0.649**
	EM	Amazon-Google	0.153	**0.657**	0.659	**0.676**	0.239	**0.566**	0.680	**0.701**
		Beer	0.500	**0.727**	0.815	**0.923**	0.741	**0.923**	0.783	**0.963**
		DBLP-ACM	0.402	**0.847**	**0.954**	0.912	0.833	**0.932**	**0.961**	0.938
		DBLP-GoogleScholar	0.206	**0.861**	0.809	**0.896**	0.632	**0.912**	0.823	**0.924**
		Fodors-Zagats	0.083	**0.872**	0.872	**0.977**	0.809	**1.000**	0.872	**0.977**
		Walmart-Amazon	0.268	**0.691**	0.519	**0.711**	0.206	**0.678**	0.664	**0.824**
		iTunes-Amazon	0	**0.788**	0.826	**0.943**	0.393	**0.862**	0.833	**0.929**
	ED	Spreadsheets-ED-Real	0.058	**0.565**	0.319	**0.552**	0.058	**0.545**	0.444	**0.551**
		WebTables-ED-Real	0.077	**0.643**	0.338	**0.545**	0.078	**0.656**	0.365	**0.685**
	SM	DeepM	1	1	1	1	0.857	**1**	1	1
	R2R	BingQL-Unit	N.A.		0.202	**0.404**	N.A.		0.333	**0.424**
		BingQL-Other			0.431	**0.588**			0.559	**0.608**
		FF-GR-Trifacta			0.716	**0.791**			0.776	**0.828**
		Headcase			0.622	**0.711**			0.689	**0.800**
		Stackoverflow			0.662	**0.745**			**0.800**	0.759

6.2 Quality Comparisons: Unseen + Seen tasks

In Figure 8, and Figure 9, we compare the performance between (GPT-3.5 vs. Table-GPT-3.5), and (ChatGPT vs. Table-ChatGPT), respectively, to see the benefit of table-tuning. There are 4 bars in each task-group, where the first two correspond to zero-shot settings, and the last two correspond to few-shot settings. We can see that across the board, table-tuned models show strong gains. Note that this benefit is observed when both GPT-3.5 and ChatGPT are used as base-models, showing the generality of table-tuning on different types of language models (completion vs. chat).

Table 3 shows a detailed breakdown of the results, at individual data-set levels. We can see that across 27 test datasets, on 2 base-models (GPT-3.5 and ChatGPT), in 2 settings (zero-shot [2] and few-shot), for a total of 98 tests in Table 3, table-tuned models outperform their vanilla counterparts in 92 out of 98 tests (with the remaining being 3 ties and 3 losses), confirming its benefits.

6.3 Table-GPT as table foundation model: benefits in downstream uses

Like discussed in Section 4.4, in addition to showing benefits in out-of-the-box zero-shot and (random) few-shot settings, table-tuned GPT models can potentially be used as "table foundation models", if they continue to show quality benefits on downstream tasks, with (1) single-task prompt-engineering, and (2) single-task fine-tuning.

[2] Zero-shot setting is not applicable to row-to-row by-example transformations, given the by-example nature of the task.

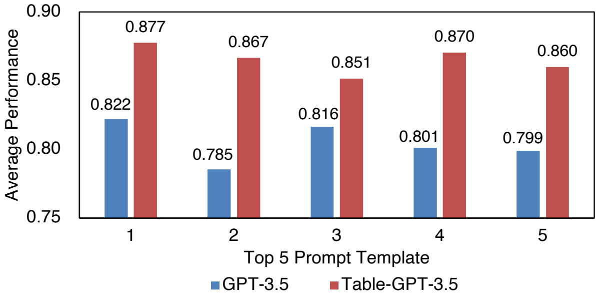
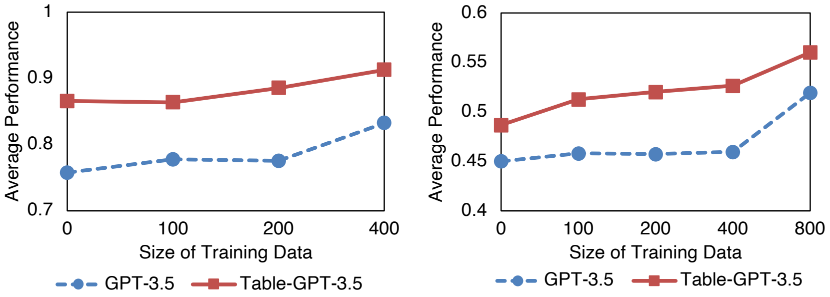


Fig. 10. Single-task prompt-engineering: quality comparison of 5 best prompt-templates (Efthymiou dataset).

Single-task prompt-engineering: We perform prompt-engineering for Table-GPT-3.5 and GPT-3.5, on the column-type-annotation (CTA) task unseen during table-tuning (using the Efthymiou [21] dataset), by selecting the best few-shot examples using 200 labeled examples (randomly sampled from the ground-truth). Figure 10 shows that for the top-5 engineered prompts, Table-GPT-3.5 consistently outperforms GPT-3.5 on all 5 prompts.

Single-task continuous fine-tuning: We perform task-specific continuous fine-tuning, on Table-GPT-3.5 and GPT-3.5, using labeled data for a specific task. Table 11(a) and (b) show the comparison on CTA (using Efthymiou [21]) and Table-QA (using WikiTableQuestions [55]), respectively, both of which are unseen tasks during training. In both cases, we vary the amount of training data on the x-axis. As expected, the performance of both Table-GPT-3.5 and GPT-3.5 benefit from fine-tuning with more task-specific labels, but with the same amount of labeled data, Table-GPT-3.5 continues to dominate GPT-3.5. Looking from the perspective of y-axis, to achieve the same performance, fine-tuning Table-GPT-3.5 requires a smaller number of labels than fine-tuning the vanilla GPT-3.5.



(a) CTA (Efthymiou) (b) TQA (WikiTableQuestion)

Fig. 11. Single-task continuous fine-tuning: quality comparison using varying amount of training data.

6.4 Sensitivity Analysis

We perform various sensitivity analysis to better understand the effect of table-tuning.

Varying the number of training tasks. To see whether using more training tasks helps, we sample 1/5/10 tasks from all of our training tasks, to perform table-tuning on each subset. We repeat the process 4 times, and report the average from the 4 in Figure 12. As we can see, with a small number of tasks (e.g., 1), table-tuning degenerates to single-task tuning, which actually hurts the performance of other tasks (the performance of Table-GPT-3.5 with 1-task is lower than that of vanilla GPT-3.5). Having more training-tasks, consistently improves overall model performance for all tasks.

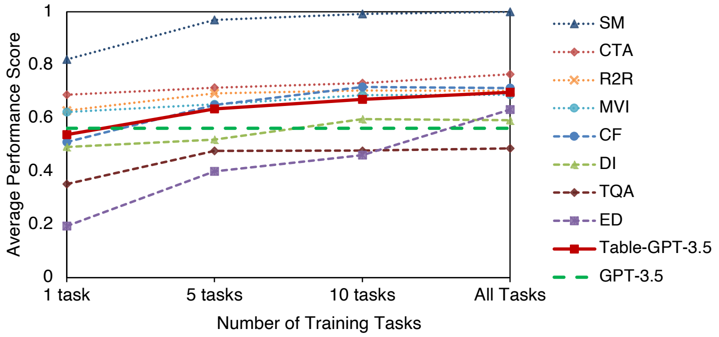


Fig. 12. Vary number of training tasks. The red solid line and the green dashed line show the average performance of Table-GPT-3.5 and GPT-3.5 over all tasks, respectively. Other lines show the breakdown of Table-GPT-3.5's performance on individual tasks, which are all generally improved with more training tasks.

Vary the amount of training data. Figure 13 shows the average performance on seen/unseen tasks with different amounts of training data (where by default, we use 1000 table-task instances per task-type). Table-GPT-3.5 improves with more training data on both seen and unseen tasks.

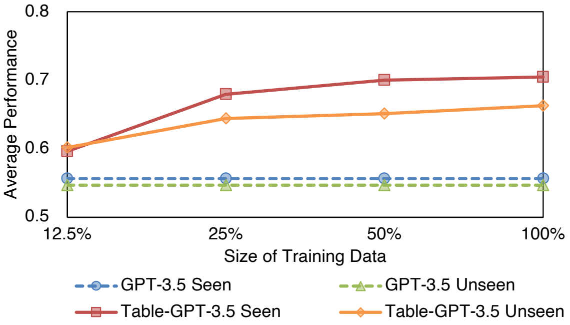


Fig. 13. Vary Training Size

Vary base-model Size. To understand how the size of the base-models affects table-tuning, we use four variants of GPT, namely, `text-ada-001` (350M parameters), `text-babbage-001` (3B parameters), `text-curie-001 (13B parameters)`, and `text-davinci-002` (175B parameters), as base models. Figure 14 shows the average performance of base-models vs. corresponding table-tuned models, on seen/unseen tasks. We can see that on smaller models (Ada/Babbage/Curie), table-tuned models produce little benefit on unseen tasks, which however becomes much more significant on larger 175B models. The ability to generalize to new tasks appears to be an ability that emerges only on large models, consistent with similar observations in other contexts (e.g., [11, 78]).

Vary prompt templates. To test the robustness of our table-tuned models, for each unseen task, we generate 5 different prompt variants (with different task descriptions, paraphrased using GPT from a human-written instruction). Figure 15 shows the average performance over all unseen test tasks for each prompt variant. While we see variations in performance with different prompts for both Table-GPT-3.5 and GPT-3.5, Table-GPT-3.5 consistently outperforms the latter by more than 10 percentage points on all 5 variants, showing the robustness of Table-GPT to different prompts.

Vary table formats. There are multiple options to serialize a table into text, such as Markdown, CSV, JSON, etc. We use the Markdown table format by default, because it is succinct, and GPT-like models seem to prefer this format when generating a table response (likely because GPT is

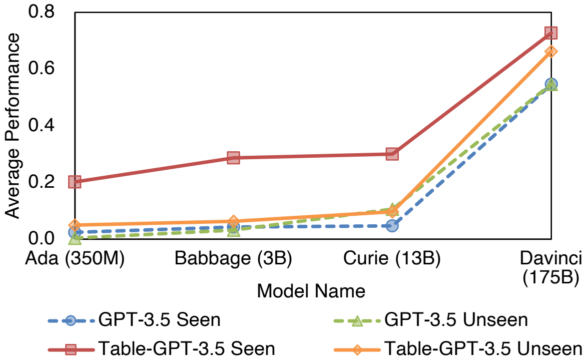


Fig. 14. Vary Model Size

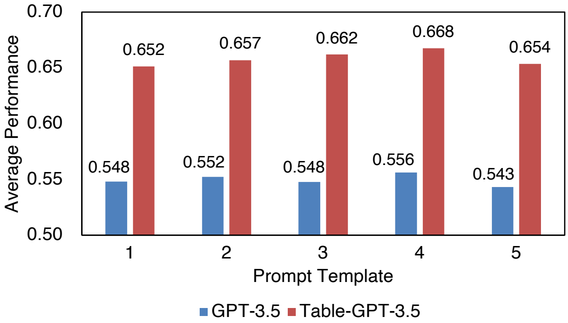


Fig. 15. Vary Templates

pre-trained on GitHub data, where Markdown tables are abundant). To understand the effect of different table formats, we test table-tuning with two additional table formats, CSV and JSON. Table 4 shows the average performance with different table formats, where Markdown performs better on average.

Table 4. Quality of Table-GPT-3.5 with different table formats

Task Type	Markdown	CSV	JSON
Seen (tasks used in training)	**0.739**	0.707	0.713
Unseen (tasks not used in training)	**0.663**	0.662	0.621
Overall	**0.705**	0.687	0.672

6.5 Ablation Studies

We perform ablation studies to understand the effect of different augmentation strategies (Section 4.3), which we report in Table 5.

Table 5. Ablation studies of table-tuning

Task Type	GPT-3.5	Table-GPT-3.5	NoSyn	NoColPer.	NoPromptVar.	NoCOT
Seen	0.548	0.739	0.610	0.735	0.722	0.728
Unseen	0.547	0.663	0.607	0.661	0.657	0.666
Overall	0.548	0.705	0.608	0.702	0.693	0.701

No task-level augmentation (no synthesized tasks). Because we synthesized diverse table-tasks for table-tuning (Section 4.2), our first ablation is to remove all such tasks from the training data. The result is shown in Table 5 as “NoSyn". As we can see, the average performance on seen and unseen tasks drops substantially, showing the contribution of the diverse tasks we synthesize.

Table 6. Performance on NLP tasks using the GLUE benchmark, with and without table fine-tuning

Task	Zero-Shot		Few-Shot		Zero-Shot		Few-Shot	
	GPT-3.5	Table-GPT-3.5	GPT-3.5	Table-GPT-3.5	ChatGPT	Table-ChatGPT	ChatGPT	Table-ChatGPT
cola	0.686	**0.810**	0.608	**0.716**	**0.785**	0.756	0.808	**0.824**
mnli_matched	0.698	**0.725**	0.763	**0.784**	0.743	**0.771**	**0.824**	0.817
mnli_mismatched	0.693	**0.718**	0.764	**0.776**	0.715	**0.761**	**0.812**	0.810
mrpc	0.725	**0.779**	0.699	**0.740**	**0.770**	0.752	0.748	**0.770**
qnli	**0.235**	0.181	0.301	**0.322**	0.136	**0.149**	**0.220**	0.202
qqp	**0.796**	0.795	0.817	**0.815**	**0.818**	0.785	0.812	**0.840**
rte	0.733	**0.787**	0.833	**0.848**	**0.866**	0.834	**0.889**	0.846
sst2	0.922	**0.933**	0.948	**0.953**	0.919	**0.929**	0.955	**0.957**
wnli	0.493	**0.507**	**0.709**	0.671	0.549	**0.592**	0.822	**0.831**

No table-level augmentation (no column permutations). We remove the table-level augmentations by turning off column permutations. The result is shown as “NoColPer". We can see that the average performance on seen and unseen tasks is lower, when this augmentation is disabled.

No instruction-level augmentation (no prompt variations). We then remove the instruction-level augmentations, by using only one canonical prompt template for each task (without paraphrasing). The result is shown as “NoPromptVar". We can see that the average performance of seen and unseen tasks drops slightly, likely because the diverse types of table-tasks we include in table-tuning, can mitigate the negative effect of using single instruction templates.

No completion-level augmentation (no chain-of-thought completion). We drop the reasoning-based augmentation (e.g., COT) at the completion-level from the training data. The result is shown as “NoCOT", which leads to lower performance on seen tasks.

Additional results. In the interest of space, we report additional experiment results, such as comparisons with existing table models, in our technical report [7].

6.6 Table-GPT on classical NLP tasks

To understand whether table fine-tuning may affect/degrade models' performance on classical NLP tasks, we evaluate both the table-tuned and vanilla models using 9 NLP datasets from the GLUE benchmark [72]. Since the labels of some test datasets are not publicly available, we use the validation sets as the test sets for all tasks.[3] Table 6 shows the classification accuracy of table-tuned and vanilla models on different tasks. For few-shot setting, we report the average accuracy over 3 different trials. As we can see, Table-GPT-3.5 generally improves GPT-3.5 on NLP tasks after table fine-tuning, even though NLP training tasks are not directly used in our training. Our hypothesis is that the models' ability of understanding instructions is improved after table fine-tuning, thereby benefiting the NLP tasks. For Table-ChatGPT, the models' performance is mixed relative to ChatGPT. We hypothesize that ChatGPT is already well-tuned to understand instructions, limiting the improvement here.

7 CONCLUSIONS AND FUTURE WORK

In this work, we propose a new paradigm called “table fine-tuning”, to continue to fine-tune large language-models like GPT-3.5 and ChatGPT, such that the resulting models are better in understanding tables and performing table tasks, while still being versatile in following diverse human instructions for unseen tasks. Just like how instruction-tuning has turned into a rich and fruitful line of research in the NLP literature, we hope our initial steps in table-tuning can serve as a springboard for new research and more optimized models in this direction.

[3] Although the original GLUE benchmark has 11 datasets, the Diagnostics Main (ax) dataset does not have a labeled test/val set and the Semantic Textual Similarity Benchmark (sstb) dataset is a regression task, both of we omit in our evaluation.

REFERENCES

[1] [n. d.]. Deepmatcher datasets. https://github.com/anhaidgroup/deepmatcher/blob/master/Datasets.md.

[2] [n. d.]. Markdown table format (GitHub). https://docs.github.com/en/get-started/writing-on-github/working-with-advanced-formatting/organizing-information-with-tables.

[3] [n. d.]. OpenAI: ChatGPT. https://openai.com/blog/chatgpt.

[4] [n. d.]. Python typo generator. https://pypi.org/project/typo/.

[5] [n. d.]. Stanford Alpaca. https://github.com/tatsu-lab/stanford_alpaca.

[6] [n. d.]. Table-GPT: Code and Data. https://github.com/microsoft/Table-GPT.

[7] [n. d.]. Table-GPT: Table-tuned GPT for diverse table tasks (Extended Version). https://arxiv.org/abs/2310.09263.

[8] Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403* (2023).

[9] Simran Arora, Avanika Narayan, Mayee F Chen, Laurel J Orr, Neel Guha, Kush Bhatia, Ines Chami, Frederic Sala, and Christopher Ré. 2022. Ask me anything: A simple strategy for prompting language models. *arXiv preprint arXiv:2210.02441* (2022).

[10] Felix Biessmann, Tammo Rukat, Philipp Schmidt, Prathik Naidu, Sebastian Schelter, Andrey Taptunov, Dustin Lange, and David Salinas. 2019. DataWig: Missing Value Imputation for Tables. *J. Mach. Learn. Res.* 20, 175 (2019), 1–6.

[11] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[12] Michael J Cafarella, Alon Y Halevy, Yang Zhang, Daisy Zhe Wang, and Eugene Wu. 2008. Uncovering the Relational Web.. In *WebDB*. Citeseer, 1–6.

[13] Jieying Chen, Jia-Yu Pan, Christos Faloutsos, and Spiros Papadimitriou. 2013. TSum: fast, principled table summarization. In *Proceedings of the Seventh International Workshop on Data Mining for Online Advertising*. 1–9.

[14] Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2019. Tabfact: A large-scale dataset for table-based fact verification. *arXiv preprint arXiv:1909.02164* (2019).

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

[16] Xu Chu, Yeye He, Kaushik Chakrabarti, and Kris Ganjam. 2015. Tegra: Table extraction by global record alignment. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. 1713–1728.

[17] Xu Chu, Ihab F Ilyas, Sanjay Krishnan, and Jiannan Wang. 2016. Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 international conference on management of data*. 2201–2206.

[18] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416* (2022).

[19] Arash Dargahi Nobari and Davood Rafiei. 2024. DTT: An Example-Driven Tabular Transformer for Joinability by Leveraging Large Language Models. *Proceedings of the ACM on Management of Data* 2, 1 (2024), 1–24.

[20] Sanjib Das, AnHai Doan, Paul Suganthan G. C., Chaitanya Gokhale, Pradap Konda, Yash Govind, and Derek Paulsen. [n. d.]. The Magellan Data Repository. https://sites.google.com/site/anhaidgroup/useful-stuff/the-magellan-data-repository?authuser=0.

[21] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2022. Turl: Table understanding through representation learning. *ACM SIGMOD Record* 51, 1 (2022), 33–40.

[22] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[23] Till Döhmen, Hannes Mühleisen, and Peter Boncz. 2017. Multi-hypothesis CSV parsing. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*. 1–12.

[24] Hazem Elmeleegy, Jayant Madhavan, and Alon Halevy. 2009. Harvesting relational tables from lists on the web. *Proceedings of the VLDB Endowment* 2, 1 (2009), 1078–1089.

[25] Raul Castro Fernandez, Aaron J Elmore, Michael J Franklin, Sanjay Krishnan, and Chenhao Tan. 2023. How Large Language Models Will Disrupt Data Management. *Proceedings of the VLDB Endowment* 16, 11 (2023), 3302–3309.

[26] Tianyu Gao, Adam Fisch, and Danqi Chen. 2020. Making pre-trained language models better few-shot learners. *arXiv preprint arXiv:2012.15723* (2020).

[27] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. 2020. Don't stop pretraining: Adapt language models to domains and tasks. *arXiv preprint arXiv:2004.10964* (2020).

[28] Braden Hancock, Hongrae Lee, and Cong Yu. 2019. Generating titles for web tables. In *The World Wide Web Conference*. 638–647.

[29] William R Harris and Sumit Gulwani. 2011. Spreadsheet table transformations from examples. *ACM SIGPLAN Notices* 46, 6 (2011), 317–328.

[30] Yeye He, Xu Chu, Kris Ganjam, Yudian Zheng, Vivek Narasayya, and Surajit Chaudhuri. 2018. Transform-data-by-example (TDE) an extensible search engine for data transformations. *Proceedings of the VLDB Endowment* 11, 10 (2018), 1165–1177.

[31] Joseph M Hellerstein. 2013. Quantitative data cleaning for large databases. (2013).

[32] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685* (2021).

[33] Zhipeng Huang and Yeye He. 2018. Auto-detect: Data-driven error detection in tables. In *Proceedings of the 2018 International Conference on Management of Data*. 1377–1392.

[34] Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César Hidalgo. 2019. Sherlock: A deep learning approach to semantic data type detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1500–1508.

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

[36] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the sigchi conference on human factors in computing systems*. 3363–3372.

[37] Moe Kayali, Anton Lykov, Ilias Fountalis, Nikolaos Vasiloglou, Dan Olteanu, and Dan Suciu. 2023. CHORUS: Foundation Models for Unified Data Discovery and Exploration. *arXiv preprint arXiv:2306.09610* (2023).

[38] Keti Korini and Christian Bizer. 2023. Column Type Annotation using ChatGPT. *arXiv preprint arXiv:2306.00745* (2023).

[39] Christos Koutras, George Siachamis, Andra Ionescu, Kyriakos Psarakis, Jerry Brons, Marios Fragkoulis, Christoph Lofi, Angela Bonifati, and Asterios Katsifodimos. 2021. Valentine: Evaluating matching techniques for dataset discovery. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 468–479.

[40] Kristina Lerman, Craig Knoblock, and Steven Minton. 2001. Automatic data extraction from lists and tables in web sources. In *IJCAI-2001 Workshop on Adaptive Text Extraction and Mining*, Vol. 98.

[41] Peng Li, Xiang Cheng, Xu Chu, Yeye He, and Surajit Chaudhuri. 2021. Auto-fuzzyjoin: Auto-program fuzzy similarity joins without labeled examples. In *Proceedings of the 2021 international conference on management of data*. 1064–1076.

[42] Peng Li, Yeye He, Cong Yan, Yue Wang, and Surajit Chauduri. 2023. Auto-tables: Synthesizing multi-step transformations to relationalize tables without using examples. *arXiv preprint arXiv:2307.14565* (2023).

[43] Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Luke Zettlemoyer, Omer Levy, Jason Weston, and Mike Lewis. 2023. Self-Alignment with Instruction Backtranslation. *arXiv preprint arXiv:2308.06259* (2023).

[44] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep entity matching with pre-trained language models. *arXiv preprint arXiv:2004.00584* (2020).

[45] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *Comput. Surveys* 55, 9 (2023), 1–35.

[46] 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 preprint arXiv:1907.11692* (2019).

[47] Weizheng Lu, Jiaming Zhang, Jing Zhang, and Yueguo Chen. 2024. Large Language Model for Table Processing: A Survey. *arXiv preprint arXiv:2402.05121* (2024).

[48] Jayant Madhavan, Philip A Bernstein, and Erhard Rahm. 2001. Generic schema matching with cupid. In *vldb*, Vol. 1. 49–58.

[49] Mohammad Mahdavi and Ziawasch Abedjan. 2020. Baran: Effective error correction via a unified context representation and transfer learning. *Proceedings of the VLDB Endowment* 13, 12 (2020), 1948–1961.

[50] Chris Mayfield, Jennifer Neville, and Sunil Prabhakar. 2010. ERACER: a database approach for statistical inference and data cleaning. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. 75–86.

[51] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 International Conference on Management of Data*. 19–34.

[52] Avanika Narayan, Ines Chami, Laurel Orr, Simran Arora, and Christopher Ré. 2022. Can foundation models wrangle your data? *arXiv preprint arXiv:2205.09911* (2022).

[53] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems* 35 (2022), 27730–27744.

[54] George Papadakis, Ekaterini Ioannou, Emanouil Thanos, and Themis Palpanas. 2021. *The four generations of entity resolution.* Springer.

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

[56] Ralph Peeters and Christian Bizer. 2023. Using ChatGPT for Entity Matching. *arXiv preprint arXiv:2305.03423* (2023).

[57] Erhard Rahm and Philip A Bernstein. 2001. A survey of approaches to automatic schema matching. *the VLDB Journal* 10 (2001), 334–350.

[58] Erhard Rahm, Hong Hai Do, et al. 2000. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.* 23, 4 (2000), 3–13.

[59] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. Holoclean: Holistic data repairs with probabilistic inference. *arXiv preprint arXiv:1702.00820* (2017).

[60] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. 2021. A primer in BERTology: What we know about how BERT works. *Transactions of the Association for Computational Linguistics* 8 (2021), 842–866.

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

[62] Ananya Singha, José Cambronero, Sumit Gulwani, Vu Le, and Chris Parnin. 2023. Tabular representation, noisy operators, and impacts on table structure understanding tasks in LLMs. *arXiv preprint arXiv:2310.10358* (2023).

[63] Jie Song and Yeye He. 2021. Auto-validate: Unsupervised data validation using data-domain patterns inferred from data lakes. In *Proceedings of the 2021 International Conference on Management of Data.* 1678–1691.

[64] Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. 2022. Annotating columns with pre-trained language models. In *Proceedings of the 2022 International Conference on Management of Data.* 1493–1503.

[65] Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. 2022. Annotating columns with pre-trained language models. In *Proceedings of the 2022 International Conference on Management of Data.* 1493–1503.

[66] Huan Sun, Hao Ma, Xiaodong He, Wen-tau Yih, Yu Su, and Xifeng Yan. 2016. Table cell search for question answering. In *Proceedings of the 25th International Conference on World Wide Web.* 771–782.

[67] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[68] Jianhong Tu, Ju Fan, Nan Tang, Peng Wang, Guoliang Li, Xiaoyong Du, Xiaofeng Jia, and Song Gao. 2023. Unicorn: A unified multi-tasking model for supporting matching tasks in data integration. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–26.

[69] Gerrit JJ van den Burg, Alfredo Nazábal, and Charles Sutton. 2019. Wrangling messy CSV files by detecting row and type patterns. *Data Mining and Knowledge Discovery* 33, 6 (2019), 1799–1820.

[70] 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 (2017).

[71] Gerardo Vitagliano, Mazhar Hameed, Lan Jiang, Lucas Reisener, Eugene Wu, and Felix Naumann. 2023. Pollock: A Data Loading Benchmark. *Proceedings of the VLDB Endowment* 16, 8 (2023), 1870–1882.

[72] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461* (2018).

[73] Pei Wang and Yeye He. 2019. Uni-detect: A unified approach to automated error detection in tables. In *Proceedings of the 2019 International Conference on Management of Data.* 811–828.

[74] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171* (2022).

[75] Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Raghavi Chandu, David Wadden, Kelsey MacMillan, Noah A Smith, Iz Beltagy, et al. 2023. How Far Can Camels Go? Exploring the State of Instruction Tuning on Open Resources. *arXiv preprint arXiv:2306.04751* (2023).

[76] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. Self-instruct: Aligning language model with self generated instructions. *arXiv preprint arXiv:2212.10560* (2022).

[77] Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Anjana Arunkumar, Arjun Ashok, Arut Selvan Dhanasekaran, Atharva Naik, David Stap, et al. 2022. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. *arXiv preprint arXiv:2204.07705* (2022).

[78] Jason Wei, Maarten Bosma, Vincent Y 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. *arXiv preprint arXiv:2109.01652* (2021).

[79] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems* 35 (2022), 24824–24837.

[80] Tim Weninger, Fabio Fumarola, Rick Barber, Jiawei Han, and Donato Malerba. 2011. Unexpected results in automatic list extraction on the web. *ACM SIGKDD Explorations Newsletter* 12, 2 (2011), 26–30.

[81] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C Schmidt. 2023. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382* (2023).

[82] Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436* (2017).

[83] Siqiao Xue, Caigao Jiang, Wenhui Shi, Fangyin Cheng, Keting Chen, Hongjun Yang, Zhiping Zhang, Jianshan He, Hongyang Zhang, Ganglin Wei, et al. 2023. Db-gpt: Empowering database interactions with private large language models. *arXiv preprint arXiv:2312.17449* (2023).

[84] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. 2012. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. 97–108.

[85] Cong Yan and Yeye He. 2018. Synthesizing type-detection logic for rich semantic data types using open-source code. In *Proceedings of the 2018 International Conference on Management of Data*. 35–50.

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

[87] 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. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Brussels, Belgium.

[88] Dan Zhang, Yoshihiko Suhara, Jinfeng Li, Madelon Hulsebos, Çağatay Demiralp, and Wang-Chiew Tan. 2019. Sato: Contextual semantic type detection in tables. *arXiv preprint arXiv:1911.06311* (2019).

[89] Haochen Zhang, Yuyang Dong, Chuan Xiao, and Masafumi Oyamada. 2023. Jellyfish: A Large Language Model for Data Preprocessing. *arXiv preprint arXiv:2312.01678* (2023).

[90] Shuo Zhang and Krisztian Balog. 2017. Entitables: Smart assistance for entity-focused tables. In *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval*. 255–264.

[91] Shuo Zhang, Zhuyun Dai, Krisztian Balog, and Jamie Callan. 2020. Summarizing and exploring tabular data in conversational search. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1537–1540.

[92] Chen Zhao and Yeye He. 2019. Auto-EM: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning. In *The World Wide Web Conference*. 2413–2424.

[93] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *CoRR* abs/1709.00103 (2017).

[94] Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. 2023. Lima: Less is more for alignment. *arXiv preprint arXiv:2305.11206* (2023).

[95] Erkang Zhu, Yeye He, and Surajit Chaudhuri. 2017. Auto-join: Joining tables by leveraging transformations. *Proceedings of the VLDB Endowment* 10, 10 (2017), 1034–1045.

Received October 2023; revised January 2024; accepted February 2024