# Generating Tables from the Parametric Knowledge of Language Models

**Yevgeni Berkovitch**[1] **Oren Glickman**[1] **Amit Somech**[1] **Tomer Wolfson**[2]
[1]Bar-Ilan University [2]Tel Aviv University
taoberkovitch@gmail.com {oren.glickman, somecha}@cs.biu.ac.il tomerwol@mail.tau.ac.il

## Abstract

We explore generating factual tables from the parametric knowledge of large language models (LLMs). While LLMs have demonstrated impressive capabilities in recreating knowledge bases and generating free-form text, their ability to generate structured tabular data has received little attention. To address this gap, we explore the table generation abilities of eight state-of-the-art LLMs, including GPT-4o and Llama3.1-405B, using three prompting methods: full-table, row-by-row, and cell-by-cell. To facilitate evaluation we introduce WikiTabGen, a new benchmark consisting of 119 manually curated Wikipedia tables and their description. Our findings show that table generation remains challenging, with the best performing model (LLaMA3.1-405B) reaching only 25.4% accuracy. We further analyze how properties like table size, popularity, and numerical content impact performance. This study highlights the unique challenges of LLM-based table generation and offers a foundation for future research in this area. All code, data, and prompts are publicly available.[1]

## 1 Introduction

Automated table generation has broad applications in fields such as healthcare, finance, scientific research and education (Chen et al., 2021; Johnson et al., 2016; Berant et al., 2018) where converting unstructured factual data into structured tables can significantly enhance decision-making, streamline workflows, and improve data accessibility enabling knowledge extraction and facilitating further analysis through statistical and visualization tools (Shen et al., 2021). Large language models (LLMs) (Brown et al., 2020; Chowdhery et al., 2022; Kadavath et al., 2022; Touvron et al., 2023a) have demonstrated remarkable performance on various natural language processing tasks, including free-form text generation, knowledge retrieval, and summarization. However, despite their success in generating free-form text, LLMs face distinct challenges when tasked with producing complex structured data, and their ability to generate long and factually accurate tables from their parametric knowledge remains largely unexplored (Akhtar et al., 2024; Zhao et al., 2024).

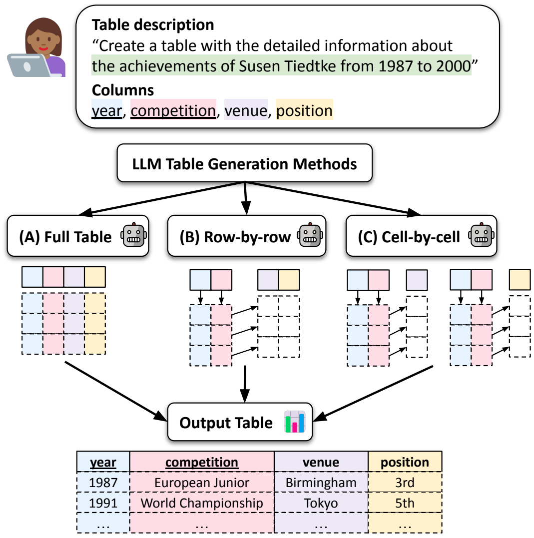



Figure 1: An example LLM-based table generation task along with three alternative prompting methods.

LLMs are pre-trained on vast amounts of text, which includes factual information presented in both plain text and structured formats, such as tables (Elazar et al., 2023; Fang et al., 2024). Through this training, LLMs encode a wealth of factual information in their parameters. While previous studies have shown that LLMs can retrieve factual information for tasks like recreating knowledge bases (KBs) (Petroni et al., 2019; AlKhamissi et al., 2022; Cohen et al., 2023) or generating Wikipedia-like articles (Shao et al., 2024), little attention has been given to their ability to generate structured tables from their parametric knowledge. Unlike question answering over tables or text-to-SQL translation (Pasupat and Liang, 2015;

[1]https://github.com/analysis-bots/WikiTabGen



*Proceedings of the 4th International Workshop on Knowledge-Augmented Methods for Natural Language Processing (KnowledgeNLP'25)*, pages 50–65
May 3, 2025 

Chen et al., 2021), generating tables requires models to retrieve and organize specific factual data into structured formats, posing unique challenges. The lack of dedicated methods for table generation and appropriate evaluation benchmarks highlights a particular gap in current research.

To address this gap, we introduce WIKITABGEN, a benchmark designed to evaluate LLMs' ability to generate tables from their parametric knowledge. It consists of 119 manually curated Wikipedia tables, each paired with a textual description and a set of target columns. With an average of 1,457 tokens per table, WIKITABGEN features significantly larger tables compared to previous tabular generation tasks (Parikh et al., 2020; Nan et al., 2022). This benchmark facilitates a systematic evaluation of how factors such as table size, numerical content, and popularity (Mallen et al., 2022) affect table generation. We also introduce and evaluate three prompting methods: full-table generation, row-by-row generation and cell-by-cell generation.

Our key contributions are: (1) Formulating the problem of generating structured tables from LLMs' parametric knowledge. (2) Introducing WIKITABGEN, a benchmark consisting of diverse tables that vary in size, structure, and content, to evaluate table generation capabilities. (3) Implementing and evaluating three prompting methods across eight state-of-the-art LLMs, including GPT-4 and LLaMA3.1-405B. (4) Providing a comprehensive analysis of the factors that impact table generation performance.

Our experiments reveal that generating tables from LLMs remains a challenging task, with the highest F1 score reaching only 25.4%. We observe that factors such as table size and numerical content significantly affect performance. These findings highlight the need for further research to improve LLM-based table generation. We hope that our benchmark and analysis will inspire future research on generating structured data from LLMs.

## 2 Problem Definition

Given a short user description, our task is to generate a factually accurate table.

Following Codd (1990), a relational table $T = (R, C)$ is a set of rows $R = \{r_1, r_2, \dots\}$ and a set of columns $C = \{c_1, c_2 \dots\}$. A table cell, denoted $r[c]$, contains the value of column $c$ in row $r$. Key columns are a subset $C_k \subset C$ that uniquely define each entry (row) in $T$ and the corresponding cells do not contain null or empty values. For example, the table in Fig. 1 has the columns `year` and `competition` as its keys. Each table entry such as `venue`, corresponds to a unique `year`, `competition` pair.

Given a table description $d$ and a list of desired table columns $C$, our task is to generate a corresponding table $T(\hat{R}, C)$, where the generated rows $\hat{R}$ contain factually accurate information. An example problem is shown in Fig. 1, where the table description is *"Achievements of Susen Tiedke from 1987 to 2000"* and the target columns are: *year*, *competition*, *venue*, and *position*. Each of our proposed prompting methods (§3) can then be used for the LLM to generate table $T(\hat{R}, C)$, as shown in the bottom of the figure.

## 3 Prompting LLMs to Generate Tables

Given a table description and list of target columns $C$, we evaluate LLM performance on generating the corresponding table $T(\hat{R}, C)$. Our focus is on extracting the knowledge stored in the LLM, with retrieval-augmented methods (Lewis et al., 2020; Yoran et al., 2023) being orthogonal to our study.

We implement three prompting methods to generate tables, shown in Fig. 1. First, the full table method prompts the LLM to generate the table all at once. However, the output table may be quite large, with evaluation tables have 1.5K tokens on average (§4). Therefore, we also experiment with a modular prompting approach (Khot et al., 2023, 2022), where one LLM instance generates the table *keys*, and another generates either complete rows or individual cells. We refer to these two modular prompting methods as row-by-row and cell-by-cell respectively. An in-depth example of our prompting methods is provided in Fig. 2. Note that all prompts in the figure are appended with the table description and columns (prompt 1 in Fig. 2). Next, we describe each of the prompts used in our three methods. All of our prompts are listed in §A and in our public code repository.

**(a) Full-table.** Given the table description and target columns the LLM is prompted to generate all table rows. Example prompts are prompts 1 and 2.A in Fig. 2 which are both concatenated and provided as the input to the LLM.

**(b) Row-by-row.** This is a two-stage prompting method, prompting two separate instances of the LLM. First, we prompt the LLM for key generation

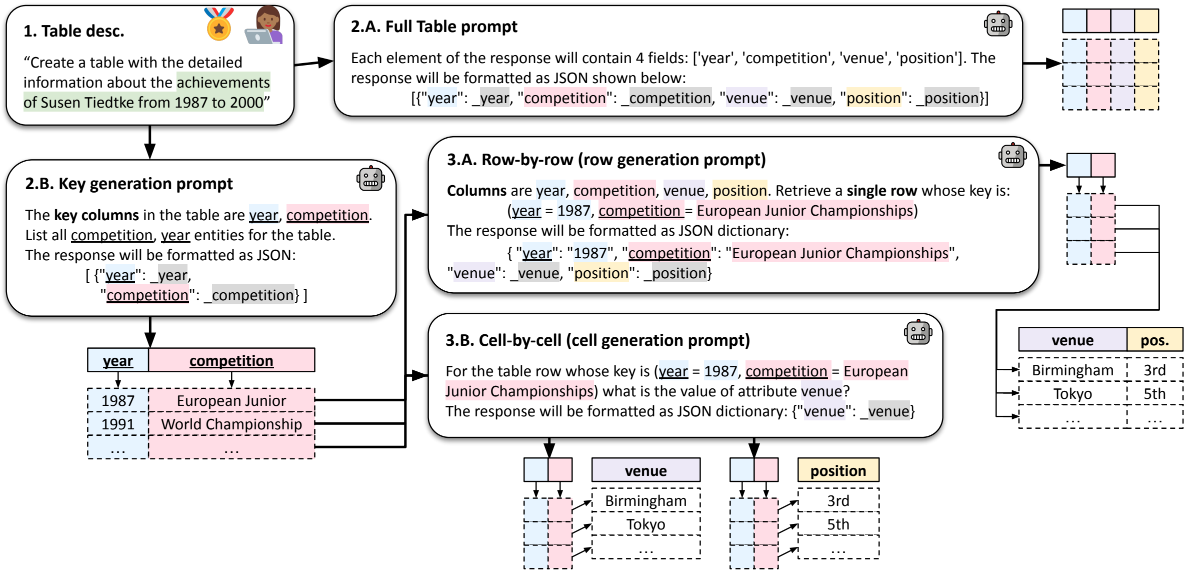



Figure 2: An overview of our three separate prompting methods for table generation, given a short user description and table metadata (Fig. 1): (2.A) Full table directly generates the table given the user desc. and its columns; (2.B) Key-generation is used in both the row-by-row and cell-by-cell methods; (3.A) Row-by-row generates a table row given a unique key value, e.g. (1987, EU Junior Championship); (3.B) Cell-by-cell generates a single table cell given a key value and specific target column e.g. *venue* $\rightarrow$ *Birmingham*.

i.e. to generate all values of the key columns $C_k$. As key values are a unique identifier for each table entry (§2), we then prompt a second instance of the LLM, to generate a full table row given a key value. Thus, for each key value $\hat{r}_i[C_k]$ generated by the first LLM, we generate a subsequent prompt to retrieve the remaining row entries $\hat{r}_i[C \setminus C_k]$. Overall, we are required to generate $|\hat{R}| + 1$ prompts, where $|\hat{R}|$ is the number of key values output by the key generation LLM.

In Fig. 2, box 2.B describes the key generation prompt. Given the table description, and key columns *competition* and *year*, the LLM generates a list of corresponding years and competitions which Susen Tiedtke participated in. Next, each key value returned by the first LLM, is used to generate the remaining row entries. Prompt 3.A prompts the row generation LLM to populate columns `venue`, `position` which correspond to key $\langle$*"European Junior"*, *"1987"*$\rangle$. The generated values being *"Birmingham"*, and "3rd". A new row-by-row prompt is then generated for the following keys, e.g. $\langle$*"World Championship"*, *"1991"*$\rangle$.

**(c) Cell-by-cell.** This two-stage approach generates each table cell individually. The first stage is identical to row-by-row, using prompt 2.B to generate all key column values. Then, we use a separate prompt for each table cell, rather than a full row. For each column $c \in C \setminus C_k$ we create a dedicated prompt to generate the cell $\hat{r}_i[c]$, based on the target column and the generated key for $r_i$. In total, we use $|\hat{R}| \cdot |C \setminus C_k| + 1$ prompts, one to generate the keys, and $|\hat{R}| \cdot |C \setminus C_k|$ to generate each of the non-key cells.

Prompt 3.B in Fig. 2 describes the cell-by-cell method. Given key $\langle$*"European Junior"*, *"1987"*$\rangle$, the corresponding cell in column `venue` is generated (*Birmingham*). The same prompt is then used for different keys and columns (`position`).

**Generated Output Format.** When prompting the LLM it is instructed to return its output in JSON format, as shown in Fig. 2. We chose JSON following past work (Singha et al., 2023) and based on our own results. Namely, we observed a better performance compared to formats such as CSV and SQL when evaluated on our held-out development set (see §4). For the row-by-row and cell-by-cell methods, we process and merge all individual JSON responses to construct the full output table.

## 4 WikiTabGen Benchmark

To evaluate our methods (§3), we introduce a new table generation benchmark called WikiTabGen. Each instance of WikiTabGen consists of a short manually written description $d$, a list of target columns $C$ and a corresponding table $T = (R, C)$. As this benchmark targets LLM table generation based on their parametric knowledge, we followed

**Susen Tiedtke (former long jumper) Achievements**

**WikiTabGen Table Meta Data:**
**Table Description:** *"Susen Tiedtke Achievement Between 1987 and 2000"*
**Key columns:** "Year", "Competition"
**Non-Key Columns:** "Venue", "Position"
**Numeric columns:** "Position" (1 of 2)
**Table size:** 10 rows, 4 columns (40 cells)
**Table Popularity:** 504.5

| Year | Competition | Venue | Position |
|---|---|---|---|
| 1987 | European Junior Championships | Birmingham, England | 3 |
| .... | .... | .... | .... |
| 1993 | World Indoor Championships | Toronto, Canada | 2 |
| 1993 | World Championships | Stuttgart, Germany | 9 |
| .... | .... | .... | .... |
| 2000 | Olympic Games | Sydney, Australia | 5 |

Figure 3: WIKITABGEN example table and metadata.

several key principles in its construction:

- Information Coverage: evaluation tables must contain complete information to prevent cases where the LLM generates correct entries that are not present in the ground-truth tables.
- Factual Consistency: tables should include static factual data, to ensure consistent evaluation over time as LLMs evolve (Zhang and Choi, 2021).
- Conciseness: table cells should contain concise string, categorical or numeric information, to avoid lengthy descriptive text that is harder to evaluate against the ground truth.
- Diversity: the benchmark should include a diverse range of tables with respect to structural properties such as size, data types (e.g., the ratio of numeric data), and table "popularity" which may indicate the prevalence of its content during the LLM's pre-training (Mallen et al., 2022).

Following these principles, we opted to use tables from Wikipedia, as our evaluation benchmark. Wikipedia is often used to assess LLMs' closed-book performance because it contains factual and objective information (Kwiatkowski et al., 2019). unlike certain domain-specific datasets (Yu et al., 2018). Additionally, since Wikipedia is part of LLMs' pre-training data (Brown et al., 2020; Touvron et al., 2023a), it is ideal for evaluating how well these models can generate tabular data.

To construct the benchmark, we iterate over the Wikipedia tables provided by Bhagavatula et al. (2015).[2] We first discarded all non-relational tables (those with composite headers, nested tables, or inverted tables) and excluded tables that were too small ($|R| < 10$ or $|C| < 2$).

Next, we manually selected 119 random tables with diverse number of columns, rows and portion of numeric values (numbers and dates). To ensure evaluation coverage we removed columns with partial entries. In addition, columns containing long texts were omitted to ensure a concise evaluation. Each table was manually annotated with a short, natural language description, as original captions were often ambiguous or not descriptive. Additionally, for tables that could change over time (e.g. new NBA championship teams), we ensured temporal specificity, as suggested by Zhang and Choi (2021), e.g. "*George Clooney Films released between 1983 and 2013*"

As shown in Fig. 3, each table in WIKITABGEN is provided with additional metadata, consisting of its: text description; table size (number of columns, rows and cells); key-columns; numeric columns (containing numbers or dates); and table popularity. Inspired by Mallen et al. (2022), we define table popularity as the average number of monthly views to the Wikipedia page containing the said table. To measure pages views we use the Wikipedia API.[3]

Overall, WIKITABGEN consists of 119 examples, with 100 used for evaluation (§5) and the remaining 19 serving as a held-out development set for method implementation. In Fig. 4 shows the distribution of three key properties in WIKITABGEN: size, numeric column ratio and popularity. On average, the evaluation tables have 77.5 rows, 6.9 columns and 453 cells, with an average length of 1,497 tokens. The average proportion of numeric columns per table is 62% of columns, showcasing the prevalence of numerical data in our tables. The average number of monthly views per table is 8,449. In §6 we further explore the effects of these properties on table generation performance.

## 5 Experimental Setting

We describe our experimental setting for evaluating the table generation capabilities of LLMs. All models were evaluated on the WIKITABGEN benchmark. Next, we list the LLMs, prompts and evaluation methods used for table generation. Last, we detail our different experimental scenarios.

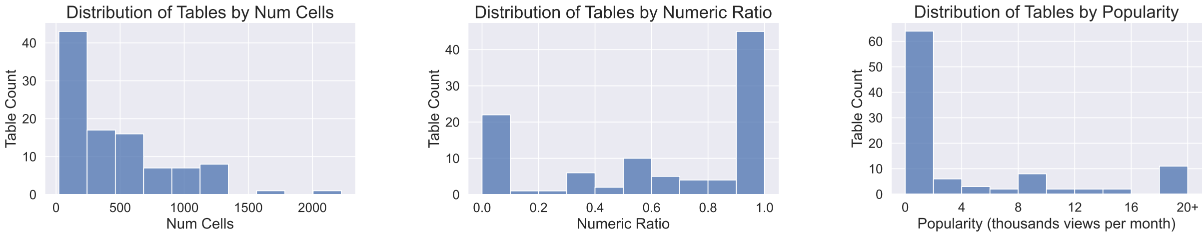
### 5.1 Language Models and Prompts

In our experiments we evaluate 8 popular LLMs: three closed-weights models by OpenAI (Achiam et al., 2023): GPT-4o, GPT-4-Turbo, GPT3.5; four open-weights LLMs by MetaAI (Touvron et al., 2023b): Llama3.1-405B, Llama3.1-70B, Llama2-70B, and Llama2-13B; and Gemma2-27B, an open-weights LLM by Google (Riviere et al., 2024).

The same prompting methods described in §3 were used across all LLMs, whereas prompt-

[2] Creative Commons Attribution 4.0 International License.

[3] `https://api.wikimedia.org`




(a) Number of cells per table

(b) Numeric columns ratio

(c) Table popularity

Figure 4: WIKITABGEN properties distribution: number of cells, ratio of numeric columns, and table popularity.

engineering was done specifically for each model, using the held-out development set as described in §4. For all LLMs, we set the generation temperature to zero.

### 5.2 Evaluation Methods

Since the order of the rows and columns in the generated table may not match the ground truth, we use the following two-step process to evaluate the generation accuracy: We align rows by key attributes, then match non-key cells.

In more detail, given output table $\hat{T}(\hat{R}, C)$ and ground-truth table $T(R, C)$, we first align the rows $\hat{R}$ to their corresponding rows in $R$ by matching their respective keys, namely $\hat{r} \rightsquigarrow r \iff \hat{r}[C_k] = r[C_k]$. For rows with multiple key columns, all values must be identical.

We then use two methods to evaluate the accuracy of cell content: *(1) exact matching*, in which we check for exact match for string content, but allow for a $\pm 0.1\%$ error for numeric content (in §B, we describe how we compare date values and handle null, missing and duplicate cells). *(2) semantic matching*, in which we first apply text-embedding on the generated and gold cell tokens, then compute the cosine similarity between them. We chose a threshold of 0.5 as our criteria for determining whether the two cells are semantically aligned.

For both matching methods we then calculate Table Precision as $\frac{\text{\# Correct Cells}}{\text{\# Generated Cells}}$, and Table Recall as $\frac{\text{\# Correct Cells}}{\text{\# Ground-Truth Cells}}$ and corresponding F1 score.

For our analysis in §6, we also consider the precision, recall, and F1 scores separately for *keys* and *non-keys*. The *keys* scores are calculated based on the number of matching keys, where for each row all the cells of $C_k$ must match. For non-key cell scores we consider only cells in $C \setminus C_k$. We provide the full formulas in Appendix B.

### 5.3 Table Generation Scenarios

In addition to the table generation scenario described in §2, where the generation request contains only the table description and list of columns, we considered two alternative scenarios where additional information is provided to the LLM:

**Table Row Example.** In this scenario, in addition to the description and list of columns, we also provide the LLM with an example row $r[C]$ from the target table. We examine if such an example improves the LLM's performance in generating the rest of the table. We tested this scenario on all prompting methods (§3) by concatenating the first row of the target table to the table description.

**Oracle Keys.** This ablation provides the LLM the ground-truth set of keys cells $R[C_k]$ and measures the model's performance in generating the remaining cells. This scenario is particularly relevant for applications where the keys are known in advance, and the task involves filling in the associated data. We conducted this experiment for both the row-by-row and cell-by-cell prompting methods by skipping the keys generation prompt (prompt 2.B in Fig. 2), and providing the ground-truth keys instead.

## 6 Results and Analysis

Following, we summarize results obtained by the 8 LLMs, 3 prompting methods and two evaluation metrics. We then analyze the generation cost and accuracy trade-offs of the prompting methods. Next, we discuss the table generation performance in our additional scenarios: *example row* and *oracle keys*, and finally, examine the effect of table properties on the LLM generation performance.

### 6.1 Main Results

Tab. 1 provides a comparison of the overall F1 scores for the eight LLMs highlighting the best performing prompting method for each model (using

| LLM | Method | Overall F1 (%) | |
|---|---|---|---|
| | | Exact | Semantic |
| LLaMa3.1-405B | Full table | **23.4** | **25.4** |
| GPT-4o | Row-by-row | 20.8 | 23.1 |
| LLaMa3.1-70B | Full table | 20.0 | 22.1 |
| GPT4-Turbo | Row-by-row | 18.9 | 21.6 |
| GPT3.5-Turbo | Full table | 16.1 | 18.0 |
| LLaMa2-70b | Row-by-row | 9.4 | 10.5 |
| Gemma2-27B | Row-by-row | 7.6 | 8.4 |
| LLaMa2-13b | Full table | 7.5 | 8.4 |

Table 1: Ranking of 8 different LLMs based on their overall F1 score (for both exact and semantic matching of tabe cells). For each LLM we only list only its best performing method.

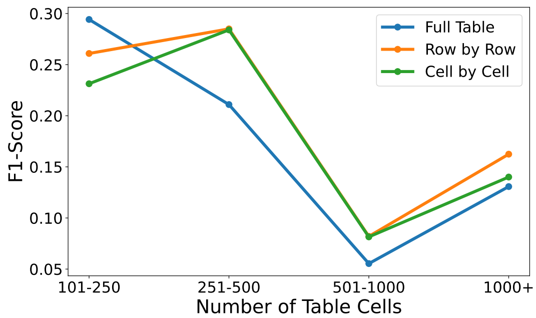



Figure 5: The performance of each prompting method for LLaMa3.1-405B with respect to the table size.

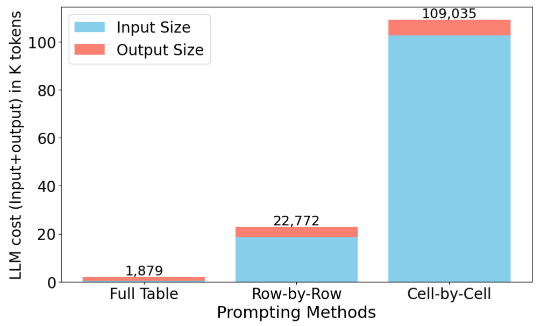



Figure 6: Cost analysis of our prompting methods.

both the exact and semantic evaluation). The top-performing model is LLaMa3.1-405B (full-table), achieving 23.4% and 25.4% F1 using the exact and semantic evaluation respectively.

We note that across all models, the semantic and exact scores are highly correlated, (semantic matching typically being approximately 10% higher than the exact score). We focus through the rest of this section on the semantic evaluation, and the top-4 performing models.

Next, Tab. 2 provides a breakdown of the performance results of the top-4 models. We list the precision, recall, and F1 scores for keys, non-keys, and the full tables (averaged across all tables), obtained for each model and prompting method.

For all LLMs, we observe that the row-by-row and cell-by-cell methods significantly improve the *keys* generation performance (see keys F1 scores in Tab. 2). Interestingly, for the two LLaMa models best performance is obtained with the full-table method, whereas for the GPT models row-by-row prompting obtained better results. Also, observe that the key generation performance is about 3X better than the non-keys, for all models. This demonstrate the inherent difficulty of current LLMs in accurately retrieving the "data" for tabular entities (as identified by the key attributes).

### 6.2 Prompting Cost Tradeoff

We analyze the performance of our prompting methods as a function of their accuracy and cost. As the row-by-row and cell-by-cell methods are suggested to handle larger tables. In Fig. 5 we examine their performance compared to the full-table method, focusing on tables with 100 or more cells on the best performing model, LLaMa3.1-405B. For medium-sized tables (100-250 cells), full-table still outperforms row-by-row. However, as the number of cells increases further, row-by-row outperforms full-table.

Next, to evaluate the cost of the prompting methods, we examine the average number of input and output tokens used for generating tables, as described in Fig. 6. While the output number of tokens is roughly similar for all approaches, see that the two-stage methods (row-by-row and cell-by-cell) have a significantly larger input due to the repeated use of distinct row and cell generation prompts (prompts 3.A, 3.B in Fig. 2).

### 6.3 Additional Generation Scenarios

We measure the effect of providing additional information during table generation: (1) an example row, (2) the ground-truth table keys.

**Table Row Example.** Tab. 3 lists the performance results when including an example row from the target table[4]. Cell-by-cell scores were omitted due to higher costs and inferior performance, as discussed in §6.2. We note that performance consistently improves when the models are given an example first row, except for GPT4-o (row-by-row), which performs slightly better given no example.

[4]As we omit the example row from the F1 calculations our

| LLM | Method | Keys | | | Non-Keys | | | Overall | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Recall | Precision | F1 | Recall | Precision | F1 | Recall | Precision | F1 |
| GPT4-Turbo | Full table | 43.4% | 66.1% | 46.8% | 12.1% | 20.6% | 13.3% | 18.0% | 28.4% | 19.4% |
| | Row-by-row | 53.9% | 57.6% | **53.2%** | 14.9% | 18.5% | **15.3%** | 21.4% | 25.0% | **21.6%** |
| | Cell-by-cell | 53.9% | 57.6% | **53.2%** | 13.5% | 17.0% | 13.8% | 20.1% | 23.6% | 20.2% |
| GPT-4o | Full table | 35.8% | 66.0% | 40.3% | 11.1% | 23.6% | 12.9% | 15.7% | 30.8% | 17.9% |
| | Row-by-row | 53.9% | 60.8% | **53.5%** | 16.3% | 21.3% | **16.8%** | 22.8% | 28.0% | **23.1%** |
| | Cell-by-cell | 53.9% | 60.7% | **53.5%** | 15.8% | 20.6% | 16.3% | 22.3% | 27.3% | 22.5% |
| LLaMa3.1-70B | Full table | 46.1% | 63.8% | 49.9% | 14.3% | 21.4% | **16.0%** | 20.1% | 28.6% | **22.1%** |
| | Row-by-row | 50.2% | 55.5% | **50.0%** | 14.3% | 16.6% | 14.3% | 20.6% | 23.3% | 20.5% |
| | Cell-by-cell | 50.2% | 55.3% | **50.0%** | 13.0% | 14.8% | 13.0% | 19.4% | 21.8% | 19.4% |
| LLaMa3.1-405B | Full table | 44.1% | 68.6% | 48.8% | 17.5% | 29.0% | **19.8%** | 22.7% | 36.0% | **25.4%** |
| | Row-by-row | 50.5% | 61.5% | **51.7%** | 15.1% | 20.4% | 15.9% | 21.2% | 27.4% | 22.1% |
| | Cell-by-cell | 50.4% | 61.4% | 51.6% | 11.8% | 15.5% | 12.3% | 18.7% | 23.7% | 19.3% |

Table 2: Table generation performance metrics for the different models and prompting methods.

| LLM | Method | Keys F1 (%) | | Non-Keys F1 (%) | | Overall F1 (%) | |
|---|---|---|---|---|---|---|---|
| | | No-Example | Example | No-Example | Example | No-Example | Example |
| GPT4-Turbo | Full table | 46.3 | **51.9** | 13.0 | **17.4** | 19.2 | **23.8** |
| | Row-by-row | 53.0 | **54.1** | 15.1 | **16.4** | 21.3 | **22.5** |
| GPT-4o | Full table | 39.7 | **47.1** | 12.6 | **16.3** | 17.7 | **22.0** |
| | Row-by-row | **53.3** | 53.3 | **16.7** | 16.5 | **22.9** | 22.8 |
| LLaMa3.1-70B | Full table | 49.4 | **51.6** | 15.5 | **18.2** | 21.6 | **24.2** |
| | Row-by-row | 49.6 | **51.6** | 14.0 | **16.6** | 20.2 | **22.5** |
| LLaMa3.1-405B | Full table | 47.9 | **50.7** | 19.2 | **25.2** | 24.7 | **29.8** |
| | Row-by-row | 51.0 | **51.9** | 15.5 | **19.9** | 21.6 | **25.6** |

Table 3: Performance comparison with and without an example row, using full table and row-by-row methods.

| LLM | Non-Keys F1 (%) | | Overall F1 (%) | |
|---|---|---|---|---|
| | Base. | Orac. | Base. | Orac. |
| GPT4-Turbo | 11.7 | 22.9 (+11.2) | 18.9 | 39.2 (+20.3) |
| GPT-4o | 13.8 | 26.1 (+12.3) | 20.8 | 41.7 (+20.9) |
| LLaMa3.1-70B | 12.2 | 25.6 (+13.4) | 19.0 | 41.4 (+22.4) |
| LLaMa3.1-405B | 14.1 | 30.9 (+16.8) | 20.7 | 45.5 (+24.8) |

Table 4: Performance comparison of the row-by-row method with and without oracle keys.

**Oracle Keys.** Tab. 4 describes the performance of all LLMs, using the row-by-row method, when given the ground-truth key values. As expected, the overall F1 scores are significantly higher when using oracle keys, because now $\hat{R}[C_k] = R[C_k]$. We observe an additional improvement in the non-keys F1, which is expected as more table rows were aligned to the target table (given the keys), and thus more cells were successfully matched.

### 6.4 Table Properties Effect on Performance

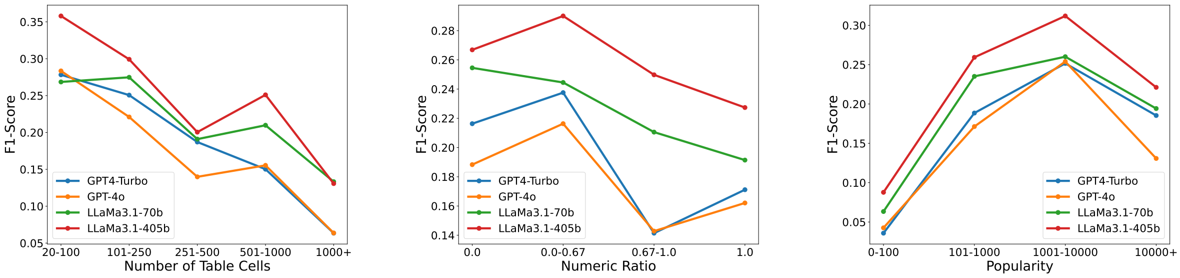
As noted in §4, we systematically measure the effect of table properties such as the size, numeric data and table popularity affect the LLM generation performance.

Fig. 7 displays the table F1 scores as a function of the number of table cells, percentage of numerical data columns (number or date cells) and the table popularity score. These results are provided for all four LLMs, using the full-table generation method. As our aim is to measure the effect each property has on the LLM (not to compare different methods). A further breakdown of the properties' effect on the keys and non-keys F1 scores is provided in Appendix §C.

As shown in Fig. 7a, the larger the table, the lower the F1 scores are for all LLMs. In §6.2 we observed this trend to be less apparent for the row-by-row and cell-by-cell methods.

Fig. 7b measures the effect the percentage of columns containing numbers or dates has on performance. We observe a general decreasing trend in F1 as the portion of numerical content is higher. Fig. 7c displays the positive effect of table popularity on performance. This potentially stems from the prevalence of more popular Wikipedia pages (or related entities) in the LLMs' training data. Unsur-

results slightly differ from Tab. 2.




(a) Table Cells Num. (b) Numeric Cols. % (c) Table Popularity Score

Figure 7: The effect of table size, the ratio of numeric columns, and table popularity on the generation performance (F1 score). The results are of the 4 top performing LLMs and using the full-table prompting method.

prising, the less common the tabular information is, the more difficult it is for the LLM to generate. We attribute the slight decrease in F1 on the top popular tables to an artifact of the data in which these tables include census related data which the LLM have difficulty to generate.

From this analysis, we conclude that generating tables from LLMs' parametric knowledge is more challenging when the tables are larger, when they contain a higher portion of numerical data and when its content concerns less popular topics.

## 7 Related Work

Machine reasoning on table using pre-trained LLMs has largely been explored in the context of data augmentation (Borisov et al., 2022; Zhang et al., 2023) to improve the performance on downstream tasks. The focus has largely been on tasks where a table is provided as input to the model namely: QA over tables (Chen et al., 2020, 2022; Seedat et al., 2023), text-to-SQL translation (Deng et al., 2021; Wolfson et al., 2022), table editing (Li et al., 2023; Sui et al., 2023) and table-to-text generation (Parikh et al., 2020). Conversely, our approach receives only a user query and schema as input, and is tasked with generating an entire table.

Closest to ours are the recent table generation datasets by Pal et al. (2023); Akhtar et al. (2024); Tang et al. (2024). In these works the LLM is provided with a user query (in text or SQL) and is tasked with generating a table, as the query answer. Pal et al. (2023) evaluate on tables from the Spider dataset (Yu et al., 2018), which contains domain-specific information that is less likely to be stored in the parametric knowledge of LLMs. In Tang et al. (2024) the authors evaluate table generation from long-form text describing NBA games, taken from the RotoWire dataset (Wiseman et al., 2017). In their setting the generated table content is already present as part of the user query, where the LLM challenge is to re-structure the user input as a table. By contrast, our setting requires the LLM to generate information that does not explicitly appear in the user input query (Fig. 1). Similar to us, Akhtar et al. (2024) rely on Wikipedia however, they automatically construct new tables which are relatively small (average of 6.7 rows, 4 columns). By comparison our evaluation is on larger tables with the median number of rows being 48 (average of 77.5 rows, 6.9 columns). This emphasizes our focus on extracting long-form tabular data from LLMs, thereby extending past attempts on KBs and text (Cohen et al., 2023; Mallen et al., 2022; Carlini et al., 2022).

Our key generation phase in §3 is an instance of a list question answering problem. The challenge of list QA in LLMs has been explored in recent works (Amouyal et al., 2022; Malaviya et al., 2023). However, we further expand this challenge by focusing on generating the entire table.

## 8 Conclusion

This paper explores the capability of state-of-the-art LLMs to generate entire tables, by relying exclusively on their parametric knowledge. We introduced three prompt-based table generation methods and evaluated them on our newly constructed benchmark, WIKITABGEN. Our results underscore the challenge table generation poses to LLMs. We hope that WIKITABGEN and our comprehensive analysis will provide a concrete framework for future research on table generation using LLMs.

## 9 Limitations

We now list the limitations to our work.

Our first limitation is the size of the WIKITABGEN evaluation benchmark, which contains 119 tables. We attribute this constraint to the intensity of the manual processing required to ensure the tables' factual correctness and robustness as well as to the high generation costs of running state-of-the-art LLMs on large tables §6.2. As noted in §4, the tables in WIKITABGEN contain close to 1,500 tokens on average, evaluating them using commercial, state-of-the-art LLMs is non-trivial.

Second, all tables in WIKITABGEN are based on Wikipedia articles. This choice was made to ensure that the underlying information exists in common LLMs training data. However, we did not examine the performance on tables generated from other sources, such as news articles or tables that require multi-source integration.

## Acknowledgements

This research was partially funded by a grant from the Data Science Institute at Bar-Ilan University.

## References

OpenAI Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mo Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Benjamin Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Sim'on Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Raphael Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Lukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Hendrik Kirchner, Jamie Ryan Kiros, Matthew Knight, Daniel Kokotajlo, Lukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Adeola Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel P. Mossing, Tong Mu, Mira Murati, Oleg Murk, David M'ely, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Ouyang Long, Cullen O'Keefe, Jakub W. Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alexandre Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Pondé de Oliveira Pinto, Michael Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario D. Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin D. Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas A. Tezak, Madeleine Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cer'on Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll L. Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2023. Gpt-4 technical report.

Mubashara Akhtar, Chenxi Pang, Andreea Marzoca, Yasemin Altun, and Julian Martin Eisenschlos. 2024. Tanq: An open domain dataset of table answered questions.

Badr AlKhamissi, Millicent Li, Asli Celikyilmaz, Mona T. Diab, and Marjan Ghazvininejad. 2022. A review on language models as knowledge bases. ArXiv, abs/2204.06031.

Samuel Joseph Amouyal, Tomer Wolfson, Ohad Rubin, Ori Yoran, Jonathan Herzig, and Jonathan Berant. 2022. Qampari: : An open-domain question answering benchmark for questions with many answers from multiple paragraphs. ArXiv, abs/2205.12665.

Jonathan Berant, Daniel Deutch, Amir Globerson, Tova Milo, and Tomer Wolfson. 2018. Explaining queries over web tables to non-experts. 2019 IEEE 35th International Conference on Data Engineering (ICDE), pages 1570–1573.

Chandra Bhagavatula, Thanapon Noraset, and Doug Downey. 2015. Tabel: Entity linking in web tables. In International Workshop on the Semantic Web.

Vadim Borisov, Kathrin Seßler, Tobias Leemann, Martin Pawelczyk, and Gjergji Kasneci. 2022. Language models are realistic tabular data generators. ArXiv, abs/2210.06280.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual.

Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramèr, and Chiyuan Zhang. 2022. Quantifying memorization across neural language models. ArXiv, abs/2202.07646.

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. 2020. Hybridqa: A dataset of multi-hop question answering over tabular and textual data. In Findings.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema N Moussa, Matthew I. Beane, Ting-Hao 'Kenneth' Huang, Bryan R. Routledge, and William Yang Wang. 2021. Finqa: A dataset of numerical reasoning over financial data. ArXiv, abs/2109.00122.

Zhiyu Chen, Shiyang Li, Charese Smiley, Zhiqiang Ma, Sameena Shah, and William Yang Wang. 2022. ConvFinQA: Exploring the chain of numerical reasoning in conversational finance question answering. In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 6279–6292, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

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

Edgar F Codd. 1990. The relational model for database management: version 2. Addison-Wesley Longman Publishing Co., Inc.

Roi Cohen, Mor Geva, Jonathan Berant, and Amir Globerson. 2023. Crawling the internal knowledge-base of language models. In Findings.

Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-grounded pretraining for text-to-SQL. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 1337–1350, Online. Association for Computational Linguistics.

Yanai Elazar, Akshita Bhagia, Ian H. Magnusson, Abhilasha Ravichander, Dustin Schwenk, Alane Suhr, Pete Walsh, Dirk Groeneveld, Luca Soldaini, Sameer Singh, Hanna Hajishirzi, Noah A. Smith, and Jesse Dodge. 2023. What's in my big data? ArXiv, abs/2310.20707.

Xi Fang, Weijie Xu, Fiona Anting Tan, Jiani Zhang, Ziqing Hu, Yanjun Qi, Scott Nickleach, Diego Socolinsky, Srinivasan Sengamedu, and Christos Faloutsos. 2024. Large language models(llms) on tabular data: Prediction, generation, and understanding - a survey. ArXiv, abs/2402.17944.

Alistair E. W. Johnson, Tom J. Pollard, Lu Shen, Li wei H. Lehman, Mengling Feng, Mohammad Mahdi Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G. Mark. 2016. Mimic-iii, a freely accessible critical care database. Scientific Data, 3.

Saurav Kadavath, Tom Conerly, Amanda Askell, T. J. Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zachary Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El-Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, John Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom B. Brown, Jack Clark, Nicholas Joseph, Benjamin Mann, Sam McCandlish, Christopher Olah, and Jared Kaplan. 2022. Language models (mostly) know what they know. ArXiv, abs/2207.05221.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2022. Decomposed prompting: A modular approach for solving complex tasks. ArXiv preprint, abs/2210.02406.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2023. Decomposed prompting: A modular approach for solving complex tasks.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: A benchmark for question answering research. Transactions of the Association for Computational Linguistics, 7:452–466.

Patrick Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. ArXiv, abs/2005.11401.

Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. 2023. Table-gpt: Table-tuned gpt for diverse table tasks. ArXiv, abs/2310.09263.

Chaitanya Malaviya, Peter Shaw, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2023. Quest: A retrieval dataset of entity-seeking queries with implicit set operations. ArXiv, abs/2305.11694.

Alex Troy Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Hannaneh Hajishirzi, and Daniel Khashabi. 2022. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In Annual Meeting of the Association for Computational Linguistics.

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

Vaishali Pal, Andrew Yates, E. Kanoulas, and M. de Rijke. 2023. Multitabqa: Generating tabular answers for multi-table question answering. In Annual Meeting of the Association for Computational Linguistics.

Ankur Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. 2020. ToTTo: A controlled table-to-text generation dataset. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1173–1186, Online. Association for Computational Linguistics.

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

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language models as knowledge bases? In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 2463–2473, Hong Kong, China. Association for Computational Linguistics.

Gemma Team Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, L'eonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ram'e, Johan Ferret, Peter Liu, Pouya Dehghani Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, Anton Tsitsulin, Nino Vieillard, Piotr Stańczyk, Sertan Girgin, Nikola Momchev, Matt Hoffman, Shantanu Thakoor, Jean-Bastien Grill, Behnam Neyshabur, Alanna Walton, Aliaksei Severyn, Alicia Parrish, Aliya Ahmad, Allen Hutchison, Alvin Abdagic, Amanda Carl, Amy Shen, Andy Brock, Andy Coenen, Anthony Laforge, Antonia Paterson, Ben Bastian, Bilal Piot, Boxi Wu, Brandon Royal, Charlie Chen, Chintu Kumar, Chris Perry, Christoper A. Welty, Christopher A. Choquette-Choo, Danila Sinopalnikov, David Weinberger, Dimple Vijaykumar, Dominika Rogozi'nska, D. Herbison, Elisa Bandy, Emma Wang, Eric Noland, Erica Moreira, Evan Senter, Evgenii Eltyshev, Francesco Visin, Gabriel Rasskin, Gary Wei, Glenn Cameron, Gus Martins, Hadi Hashemi, Hanna Klimczak-Pluci'nska, Harleen Batra, Harsh Dhand, Ivan Nardini, Jacinda Mein, Jack Zhou, James Svensson, Jeff Stanway, Jetha Chan, Jin Zhou, Joana Carrasqueira, Joana Iljazi, Jocelyn Becker, Joe Fernandez, Joost R. van Amersfoort, Josh Gordon, Josh Lipschultz, Joshua Newlan, Junsong Ji, Kareem Mohamed, Kartikeya Badola, Kat Black, Katie Millican, Keelin McDonell, Kelvin Nguyen, Kiranbir Sodhia, Kish Greene, Lars Lowe Sjoesund, Lauren Usui, L. Sifre, L. Heuermann, Leticia Lago, Lilly McNealus, Livio Baldini Soares, Logan Kilpatrick, Lucas Dixon, Luciano Martins, Machel Reid, Manvinder Singh, Mark Iverson, Martin Gorner, Mat Velloso, Mateo Wirth, Matt Davidow, Matt Miller, Matthew Rahtz, Matthew Watson, Meg Risdal, Mehran Kazemi, Michael Moynihan, Ming Zhang, Minsuk Kahng, Minwoo Park, Mofi Rahman, Mohit Khatwani, Natalie Dao, Nenshad Bardoliwalla, Nesh Devanathan, Neta Dumai, Nilay Chauhan, Oscar Wahltinez, Pankil Botarda, Parker Barnes, Paul Barham, Paul Michel, Pengchong Jin, Petko Georgiev, Phil Culliton, Pradeep Kuppala, Ramona Comanescu, Ramona Merhej, Reena Jana, Reza Rokni, Rishabh Agarwal, Ryan Mullins, Samaneh Saadat, S. Mc Carthy, Sarah Perrin, S'ebastien Arnold, Sebastian Krause, Shengyang Dai, Shruti Garg, Shruti Sheth, Sue Ronstrom, Susan Chan, Timothy Jordan, Ting Yu, Tom Eccles, Tom Hennigan, Tomás Kociský, Tulsee Doshi, Vihan Jain, Vikas Yadav, Vilobh Meshram, Vishal Dharmadhikari, Warren Barkley, Wei Wei, Wenming Ye, Woohyun Han, Woosuk Kwon, Xiang Xu, Zhe Shen, Zhitao Gong, Zichuan Wei, Victor Cotruta, Phoebe Kirk, Anand Rao, Minh Giang, Ludovic Peran, Tris Brian Warkentin, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, D. Sculley, Jeanine Banks, Anca Dragan, Slav Petrov, Oriol Vinyals, Jeffrey Dean, Demis Hassabis, Koray Kavukcuoglu, Cl'ement Farabet, Elena Buchatskaya, Sebastian Borgeaud, Noah Fiedel, Armand Joulin, Kathleen Kenealy, Robert Dadashi, and Alek Andreev. 2024. Gemma 2: Improving open language models at a practical size. ArXiv, abs/2408.00118.

Nabeel Seedat, Nicolas Huynh, Boris van Breugel, and Mihaela van der Schaar. 2023. Curated llm: Synergy of llms and data curation for tabular augmentation in ultra low-data regimes. ArXiv, abs/2312.12112.

Yijia Shao, Yucheng Jiang, Theodore A. Kanell, Peter Xu, Omar Khattab, and Monica S. Lam. 2024. Assisting in writing wikipedia-like articles from scratch with large language models. ArXiv, abs/2402.14207.

Leixian Shen, Enya Shen, Yuyu Luo, Xiaocong Yang, Xuming Hu, Xiongshuai Zhang, Zhiwei Tai, and Jianmin Wang. 2021. Towards natural language interfaces for data visualization: A survey. IEEE Transactions on Visualization and Computer Graphics, 29:3121–3144.

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.

Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2023. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. In Web Search and Data Mining.

Xiangru Tang, Yiming Zong, Jason Phang, Yilun Zhao, Wangchunshu Zhou, Arman Cohan, and Mark Gerstein. 2024. Struc-bench: Are large language models good at generating complex structured tabular data? In Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers), pages 12–34, Mexico City, Mexico. Association for Computational Linguistics.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aur'elien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models. ArXiv, abs/2302.13971.

Hugo Touvron, Louis Martin, Kevin R. Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Daniel M. Bikel, Lukas Blecher, Cristian Cantón Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony S. Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel M. Kloumann, A. V. Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, R. Subramanian, Xia Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zhengxu Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models. ArXiv, abs/2307.09288.

Sam Wiseman, Stuart Shieber, and Alexander Rush. 2017. Challenges in data-to-document generation. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 2253–2263, Copenhagen, Denmark. Association for Computational Linguistics.

Tomer Wolfson, Daniel Deutch, and Jonathan Berant. 2022. Weakly supervised text-to-SQL parsing through question decomposition. In Findings of the Association for Computational Linguistics: NAACL 2022, pages 2528–2542, Seattle, United States. Association for Computational Linguistics.

Ori Yoran, Tomer Wolfson, Ori Ram, and Jonathan Berant. 2023. Making retrieval-augmented language models robust to irrelevant context.

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, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Michael Zhang and Eunsol Choi. 2021. SituatedQA: Incorporating extra-linguistic contexts into QA. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 7371–7387, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

T. Zhang, Shaowen Wang, Shuicheng Yan, Jian Li, and Qian Liu. 2023. Generative table pre-training empowers models for tabular prediction. ArXiv, abs/2305.09696.

Yilun Zhao, Lyuhao Chen, Arman Cohan, and Chen Zhao. 2024. TaPERA: Enhancing faithfulness and interpretability in long-form table QA by content planning and execution-based reasoning. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 12824–12840, Bangkok, Thailand. Association for Computational Linguistics.

# A Table Generation Prompts

In this section we provide the prompt templates used in each of our table generation methods. Figs. 8-11 present our prompt templates used for: full table generation method, keys generation, row-by-row, and cell-by-cell method.

# B Evaluation Method Details

## B.1 Precision-Matching of Cell Values

We next describe our precision matching for cell values in more detail, given an output table $\hat{T}(\hat{R},C)$ and ground-truth table $T(R,C)$.

As described in §5.2, we use *exact* value comparison of cell textual content and allow a $\pm 0.1\%$ error for numeric values. Before comparing textual cells, we first convert them to lower case, and remove non alphanumeric symbols and spaces.

As for date values, we first parse and convert cells with date values to a Python Date object, and then compare the canonical dates. This is to avoid cases where cells are deemed as a non-match due to differences in the date format. For example, in our evaluation process, two date values representing the same date, such as "2014-05-16" and "16th, May, 2014", will be considered the same.

We further treat “none”, “n/a” ,“nan” and empty cells as identical in terms of value matching.

## B.2 Precision and Recall Computation for Tables

For a given output table $\hat{T}(\hat{R},C)$ and ground-truth table $T(R,C)$, we first align the rows $\hat{R}$ to their corresponding rows in $R$ by matching their respective keys, namely $\hat{r} \rightsquigarrow r \iff \hat{r}[C_k] = r[C_k]$. For rows with composite keys, all key values must be identical, i.e., $\forall c_k \in C_k \hat{r}[c_k] = r[c_k]$.

Recall that a *correct* cell in $T(\hat{R},C)$ is a cell $\hat{r}[c]$ such that $\hat{r}[c] \approx r[c] \wedge \hat{r} \rightsquigarrow r$. Namely, row $\hat{r}$ is aligned with a row $r$ in the ground-truth table, and their corresponding cell values in column $c$ is matching (using either the precision or semantic matching definition).

We next provide the precision and recall formulas we used for keys, non-keys, and tables.

For keys, we compare $\hat{R}[C_k]$ and $R[C_k]$ as follows. Let the number of matching keys $\phi = |\{r \in \hat{R}, \forall c_k \in C_k \; \hat{r}[c_k] = r[c_k]\}|$ Then *keys precision* is calculated by $\frac{\phi}{|\hat{R}|}$ and *keys recall* is given by $\frac{\phi}{|R|}$.

For non-keys, we compare $\hat{R}[C \setminus C_k]$ and $R[C \setminus C_k]$. After aligning $\hat{R}$ and $R$, we compute the num-

**Full-table generation template:**
You are a retriever of facts. List all `{table description}`. The response will be formatted as JSON shown below. Each element of the response will contain `{num columns}` fields: `{column1, column2, ...}`
Do not output any additional text that is not in JSON format.
RESPONSE FORMAT: [{ `column1: value1, column2: value2, ...` }]

**Full-table generation (populated example):**
You are a retriever of facts. List all achievements of Susen Tiedtke from 1987 to 2000. The response will be formatted as JSON shown below. Each element of the response will contain 4 fields: ['year', 'competition', 'venue', 'position']. Do not output any additional text that is not in JSON format.
RESPONSE FORMAT: [{ "year": _year, "competition": _competition, "venue": _venue, "position": _position }]

Figure 8: Full-table generation prompt.

**Keys generation template:**
You are a retriever of facts. We want to create a table with the detailed information about `{table description}`. The key columns in the table are `{key1, (key2, ...)}`. List all `{key1, (key2, ...)}` entities for the table. The response will be formatted as JSON list shown below.
RESPONSE FORMAT: [{ `key: value1, key2: value2, ...` }]

**Keys generation (populated example):**
You are a retriever of facts. We want to create a table with the detailed information about achievements of Susen Tiedtke from 1987 to 2000. The key columns in the table are competition, year. List all competition, year entities for the table. The response will be formatted as JSON list shown below.
RESPONSE FORMAT: [{ "competition": _competition, "year": _year }]

Figure 9: Key columns generation prompt.

ber of *correct* keys, denoted by $\psi = |\{(r,c), r \in \hat{R} \wedge c \in C \setminus C_k \wedge r \rightsquigarrow \hat{r} \wedge \hat{r}[c] = r[c]\}|$. Then the *non-keys precision* is calculated by $\frac{\psi}{|\hat{R}[C \setminus C_k]|}$ and *non-keys recall* is calculated by $\frac{\psi}{|R[C \setminus C_k]|}$.

Last, for the table precision and recall, we perform a similar evaluation, now defining the number of correct cells, denoted by $\tau$, as all correct cells in the table. Namely, $\tau = |\{(r,c), r \in \hat{R} \wedge c \in C \wedge r \rightsquigarrow \hat{r} \wedge \hat{r}[c] \approx r[c]\}|$, then the *table precision* is simply calculated by $\frac{\tau}{|\hat{R}[C]|}$ and *table recall* is calculated by $\frac{\tau}{|R[C]|}$.

## C Table Properties Effect on Performance

In §6.4 we examine how the table properties such as the size, amount of numeric data, and table popularity affect the generation performance. In Fig. 12 we present the effect of these three properties on both the keys F1, non-keys F1, and full table F1. We can see, for instance, that the table size negatively affects both the keys F1 and the non-keys F1 scores (see Fig. 12 (a) and Fig. 12 (b)), and the ratio of numeric columns has a negative effect, as expected, only the non-keys F1 (see Fig. 12 (e)). The table popularity also have a strong effect on both the keys F1 and the non-keys F1 (Fig. 12 (g) and Fig. 12 (h)).

**Row generation template:**
You are a retriever of facts. We want to create a table with the detailed information about `{table description}`. Columns in the table are `{columns}`. The key columns in the table are `{key1, (key2, ...)}`. Retrieve a single row whose key is `({key = value})`. The response will be formatted as JSON dictionary shown below. Pay special attention to wrap all values in double quotes!
RESPONSE FORMAT: [{ `column1: value1, column2: value2, ...` }]

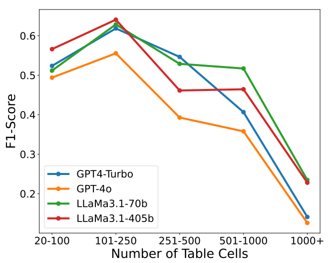
**Row generation (populated example):**
You are a retriever of facts. We want to create a table with the detailed information about achievements of Susen Tiedtke from 1987 to 2000. Columns in the table are year, competition, venue, position. The key columns in the table are competition, year. Retrieve a single row whose key is (year = 1987, competition = World Championships). The response will be formatted as JSON dictionary shown below. Pay special attention to wrap all values in double quotes!
RESPONSE FORMAT: { "year": 1987, "competition": World Championships, "venue": _venue, "position": _position }

Figure 10: Row-by-row (row generation) prompt.

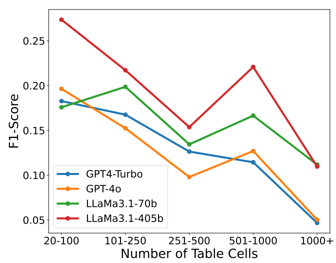
**Cell generation template:**
You are a retriever of facts. We want to create a table with the detailed information about `{table description}`. Columns in the table are {`column1`, `column2`, ...}. The key columns in the table are `{key1, (key2, ...)}`. For the table row whose key is is `({key = value})` what is the value of attribute `{column}`. The response will be formatted as JSON dictionary shown below. Pay special attention to wrap all values in double quotes!
RESPONSE FORMAT: { `column: value` }

**Cell generation (populated example):**
You are a retriever of facts. We want to create a table with the detailed information about achievements of Susen Tiedtke from 1987 to 2000. Columns in the table are year, competition, venue, position. The key columns in the table are competition, year. For the table row whose key is (year = 1987, competition = World Championships) what is the value of attribute venue. The response will be formatted as JSON dictionary shown below. Pay special attention to wrap all values in double quotes!
RESPONSE FORMAT: { "venue": _venue }

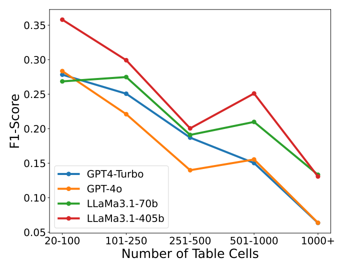
Figure 11: Cell-by-cell (cell generation) prompt.



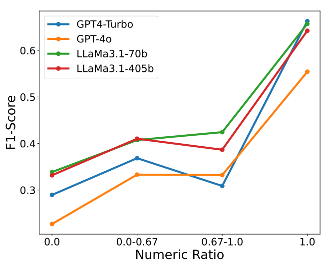

(a) Table Cells Num. - Keys F1



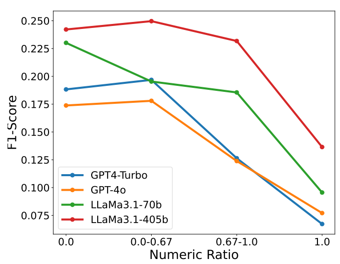

(b) Table Cells Num. - Non-Keys F1



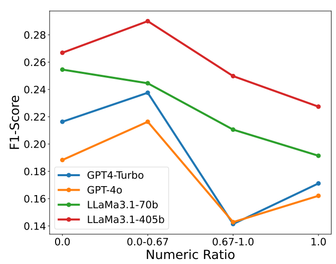

(c) Table Cells Num. - Table F1



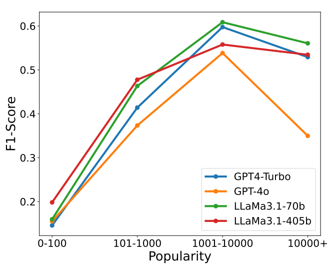

(d) Numeric Cols. % - Keys F1



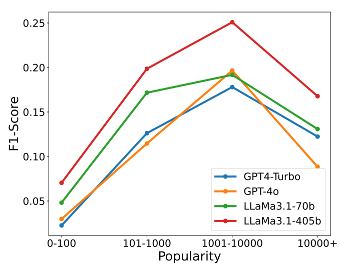

(e) Numeric Cols. % - Non-Keys F1



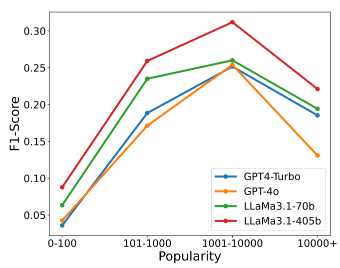

(f) Numeric Cols. % - Table F1




(g) Table Popularity - Keys F1




(h) Table Popularity - Non-Keys F1




(i) Table Popularity - Table F1

Figure 12: The effect of table size, the ratio of numeric columns, and table popularity on the generation performance of the full-table method, with four different LLMs. Additional breakdown of generation performance based on cells in key columns versus non-key columns.