A. Technical Report

A.1. Unionability Benchmark Generation

Initial Setup Before prompting the LLMs about generating the tables, we first populate the topic, shape, and textuality requirements for the table. In our case, we prompt an LLM to generate tables related to different topics in order to create diverse and realistic benchmarks. An important property, which is not present in other benchmarks, is the ability to generate non-unionable tables of the same topic and to let the user of the generator pick the number of topics (meaning how diversity they want their benchmark to be).

Generating Column Headers and the Verification Process For each topic, we have a query table and a set of data lake tables that are either unionable or not unionable to the former table. We first generate a pair of query and data lake table. As mentioned in Section 3.2, LLMs can be sensitive to the prompts provided since it can interpret the task differently with any subtle variations of the task's description. Hence, while instructing LLMs, we first describe the shape of the tables.

Create 2 semi-colon-separated table column headers. Table 1 has {$T1_{col}$} columns on the topic of {$T1_{topic}$}. Table 2 has {$T2_{col}$} columns on the same topic.

Then, we instruct the LLM about their unionability:

They can be unioned because they have {$T_{unionable\_col}$} semantically similar columns that can be aligned in both tables. In other words, {$T_{unionable\_col}$} of the columns in table 2 resemble columns in table 1. The remaining columns don't necessarily resemble any of the columns in table 1.

They cannot be unioned because there are no columns in table 2 that are semantically similar to any columns in table 1 and vice-versa. In other words, none of the columns in table 2 resemble any of the columns in table 1.

In the prompts, {$T1_{col}$} and {$T1_{topic}$} (and similarly for $T2$) as well as {$T_{unionable\_col}$} are parameters. They may be instantiated randomly during the 'initial setup' phase in Figure 1, but their ranges can also be set by users if a table-pair needs to have certain user-specific conditions. For example,{$T_{unionable\_col}$} represents the number of semantically similar columns that are present between the two tables if they are unionable. To generate benchmarks for evaluating SANTOS [7] we might set the range of this parameter to be from two to ten (since SANTOS requires a minimum of two unionable columns).

To further reduce prompt sensitivity and to allow for increased consistency in the format of the output generated by an LLM, we add the following instruction to the prompt template.

```
Answer the above task in the following
format:
Table 1: <table 1-separated table 1 column
headers>
Table 2: <table 2-separated column headers>
```

To combat another challenge of LLMs, i.e., hallucination (refer: Section 3.2, we add a verification phase, where we use the response of the LLM from the above prompt, and ask the LLM to verify its' response. Here's an instance of prompt template:

Verify that Table 1 has {$T1_{col}$} columns and Table 2 has {$T2_{col}$} columns and the generated tables can be unioned. Re-generate your answer with the corrected response in the requested format.

In the non-unionable case, we change the 'can' to 'cannot' and further clarification that 'none of the columns are semantically similar to any columns in table 1 and vice-versa.'

After verifying the initial query and data lake table pairs for a particular topic, we use the query table column headers and repeat the above process for other data lake tables as illustrated in Figure 2.

Generating Rows After storing the generated column headers for each table in the benchmark, we populate these tables by prompting the LLM to produce first T_{rows} rows for these tables given their respective column headers. To do so,

Given the following column header for a table about {topic}, generate T_{rows} table rows where each row has at least {textuality} words. Here's the column header: {tableHeaders}. Answer this task in the format of semi-separated rows, where each row is in a new line.

To address scalability, we divide up the task of adding many rows into the previous prompt of generating the first set of rows, and then prompting LLM to add more rows, using previously extracted table headers and a sample of row texts:

Given the column headers and last couple rows for a table about {topic}, generate

```
Trows more table rows where each row has at
least {textuality} words. Here's the column
header: {tableHeaders}. Here's the last
couple rows:{rowTexts}. Answer this task
in the format of semi-colon-separated rows,
where each row is in a new line.
```

An LLM outputs the requested instructions/prompts as strings. We use post-processing scripts to convert the string-formatted tables into CSV files. In cases were a union search method might require more information than the above tables, we have other scripts to further prompt the LLM and retrieve the required information. For instance, a key or intent column is required by one of the search techniques that we will evaluate [7]. For this, we prompt the LLM to provide us this column based on the query-table and data-lake table pair column headers provided to it.

A.2. Union Search Methods

We evaluate the publicly available recent union search methods over the existing and new benchmarks.

D^{3}L [4]. Bogatu et al. extended TUS [3] to use not only word embeddings, knowledge graph mappings, or value overlap, but also column header similarity, distributions for numerical columns, and regular expressions.

We used D^3L's publicly available code.[2] For a fair comparison, we do not use the column header similarity metric since the existing benchmarks use identical schema names for unionable columns [3, 7].

SANTOS [7]. SANTOS uses column semantics and the semantics of relationships between column pairs to search for the unionable tables. SANTOS only uses column values and does not use metadata like column headers. To find column semantics and relationships semantics, SANTOS uses an external knowledge base and a synthesized knowledge base created using the data lake itself. To run SANTOS, we use the public code provided with the paper.[3]

Starmie [8]. Starmie is a recent self-supervised table union search technique based on contrastive learning. Starmie captures the table context in the form of contextualized column embeddings and uses them to perform table union search. We reproduced Starmie following the instruction in its open implementation.[4]

Starmie-LLM. LLMs cannot be used directly for the search task but, as mentioned, can be used for union classification. Therefore, to assess LLMs in the table union search task, we use an existing table union search method to search for a set of candidate unionable tables for each query table. Then we prompt an LLM to classify whether the query table is unionable with each of these candidate tables. This two-phase approach is very common for information retrieval applications [48, 49] in which two models are applied consecutively. In our experimental setup, we use Starmie to search for a larger number of candidate unionable tables. Then we prompt an LLM to classify each query-candidate table pair as unionable or not by asking the following question:

```
Are the following tables unionable? Answer
in the following format: Unionable: {yes/no}
```

We use recent LLMs that have shown promising performance in other generative tasks. Specifically, we use GPT2-XL[5], Alpaca (7 Billion parameters)[6], and Vicuna (7 billion parameters)[7] in our experiments. We denote respective LLM variations using Starmie-GPT2-XL, Starmie-Alpaca, and Starmie-Vicuna.

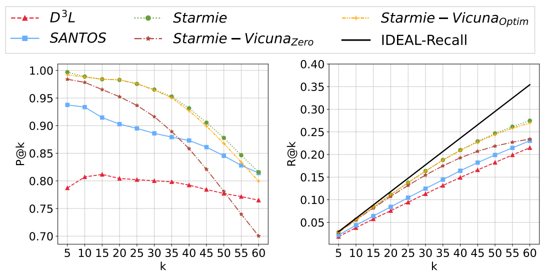
LLMs can function either as they are (zero-shot) or can be directed towards specific tasks by providing a few examples (in-context learning). For comparison with other methods, we denote zero-shot versions denoted as Starmie-LLM$_{\text{Zero}}$ and in-context versions with an optimal number of examples (meaning the best performance against other in-context versions) as Starmie-LLM$_{\text{Optim}}$.

Further results for other smaller values of k are plotted in Fig. 5 with different values of k in horizontal axes, the evaluation metrics in vertical axes, and the methods encoded using different colors and line styles. As noted, the recall cannot be perfect if k is smaller than the ground truth size [7]. So, we show the IDEAL-RECALL line that indicates the maximum possible recall for each value of k.

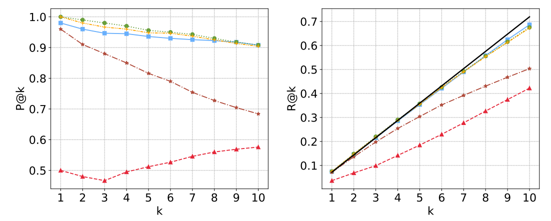
Runtime Performance In Section 3.4, we reported the time and cost (if any) to generate UGEN-V1 and UGEN-V2 benchmarks. The Starmie-LLM variation methods had varying query-time based on their model sizes, length of the prompt, and platform for running them. In the zero-shot case for UGEN-V2, Starmie-GPT-2XL took a total of about 12 minutes, Starmie-Vicuna took about 15 minutes, and Starmie-Alpaca took around 25 minutes. GPT2-XL is a 1.5 Billion parameter model while Vicuna and Alpaca models are around 7 Billion parameter models. Hence, GPT2-XL had a relatively faster inference time than Vicuna and Alpaca. These query times increase non-linearly. Furthermore, between zero-shot and three-shot, there is about 50% increase in average query time for all models. This is due to the significant increase of the prompt length from a prompt without any examples to one with 3 examples.

[2]https://github.com/alex-bogatu/d3l
[3]https://github.com/northeastern-datalab/santos
[4]https://github.com/megagonlabs/starmie

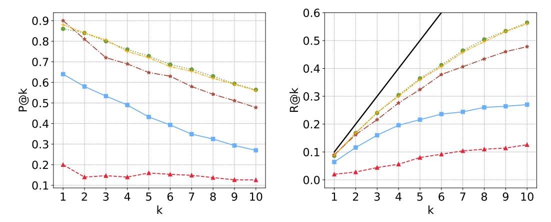
[5]https://huggingface.co/gpt2-xl
[6]https://huggingface.co/circulus/alpaca-7b
[7]https://huggingface.co/lmsys/vicuna-7b-v1.3



(a) Average $P@k$ on TUS-Small **(b)** Average $R@k$ on TUS-Small



(c) Average $P@k$ on Santos-Small **(d)** Average $R@k$ on Santos-Small

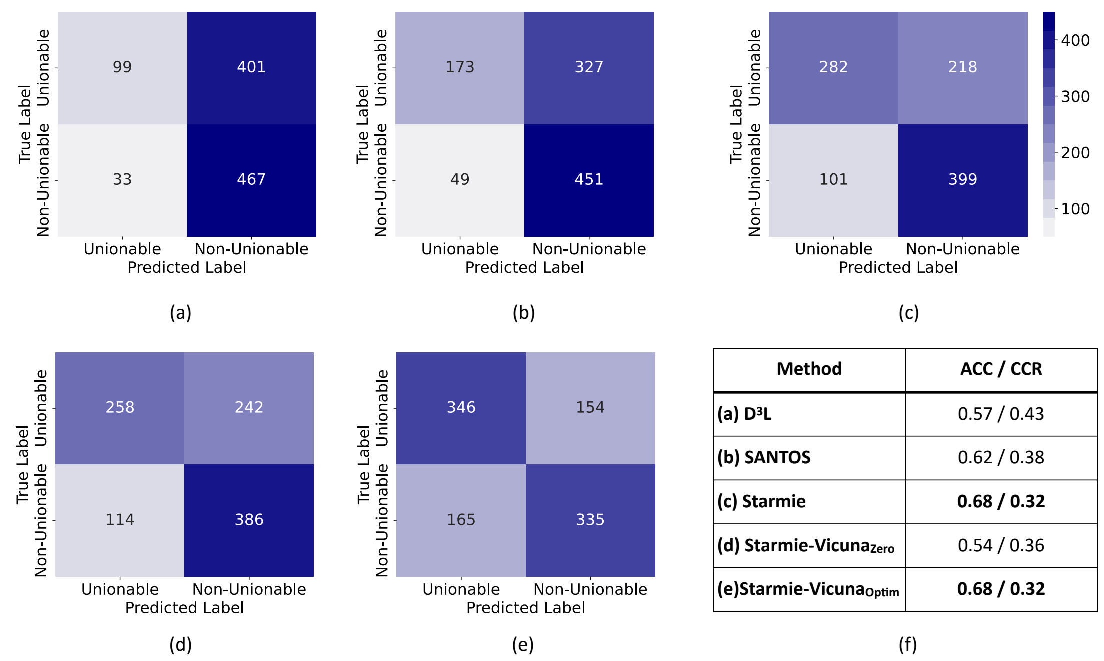


(e) Average $P@k$ on UGEN-V2 **(f)** Average $R@k$ on UGEN-V2

Figure 5: Effectiveness of baselines in different benchmarks

Finally,we also create a confusion matrix, we need true labels, false labels, and predicted labels. True labels and false labels are ”unionable” and ”non-unionable” table pairs from the ground truth respectively. Note that we use search methods that return top-k results for each query. As the total number of unionable table pairs for each query is 10 in the ground truth, we search for the top-10 unionable tables for each query. With 50 query tables, we get 500 predicted true labels. The table pairs that are in the ground truth, but do not make it to the top-10 are false (non-unionable) predicted labels. We compare predicted labels with the ground truth to construct confusion matrices.

We report confusion matrix and accuracy in Figure 6 for each method on UGEN-V2. If we see correct non-unionable predictions (bottom-right of confusion matrices), D^3L seems to be the best at detecting non-unionable tables (predicts 467 out of 500 non-unionable pairs accurately). Notice however, if we look at true positives (top-left of confusion matrix in Figure 6(a)), D^3L has the least number, around two times fewer than the second-least performing SANTOS (Figure 6 (b)). This means that the false positives in D^3L's results are other unlabeled non-unionable tables from random topics rather than the labeled ”non-unionable” tables from the same topic. But if we look at Starmie and Starmie's LLM variations that capture the table context to make unionability decisions, they have high true positives and relatively lower true negatives. This means that their false positives are mostly non-unionable tables from the same topic, as we discussed in Example 1. For instance, Starmie-$\text{Vicuna}_{\text{Optim}}$ seems to be the most balanced method in differentiating unionable and non-unionable among the same topic, achieving the highest accuracy. This may be because Starmie captures overall table context well and from the remaining unionable candidates, LLM may also be less confused on the tables from the same topic. This study does reveal that by labeling non-unionable pairs in UGEN-V2, we have been able to create difficult cases that lead to false negatives for all methods. We also get very interesting insight into table unionability – it is important that our benchmarks are not just testing if a method can find tables on the same topic. Instead, the search methods should also separate non-unionability among the same topic tables.



Method	ACC / CCR
(a) D³L	0.57 / 0.43
(b) SANTOS	0.62 / 0.38
(c) Starmie	**0.68 / 0.32**
(d) Starmie-Vicuna$_{Zero}$	0.54 / 0.36
(e)Starmie-Vicuna$_{Optim}$	**0.68 / 0.32**

Figure 6: Confusion matrices for (a) D^{3}L (b) SANTOS (c) Starmie (d) Starmie-Vicuna$_{Zero}$ (e) Starmie-Vicuna$_{Optim}$ methods on distinguishing unionable and non-unionable pairs in UGEN-V2 Benchmark. A legend in the top-right corner is for all the confusion matrices. (f) Accuracy and Corner Case Ratio of different methods in UGEN-V2 Benchmark