PROJECT REPORT FOR CS 839
Team ID: 2
Member names: Son Dam, Ethan Fang, Kefan Zheng

Towards the Use of Large Language Models for Semantic Column Type Detection

0) Abstract

When analyzing a dataset, data scientists must know the semantic type of each column before performing analysis. However, in many cases these semantic types are not provided, creating a tedious process for a human to go through each column and manually label it. Our study looks into how well Meta's LLaMA Large Language Model performs on this task.

1) Problem Description

When analyzing a dataset, a data scientist must determine the semantic type of each column to know how to interpret it and decide what type of analysis to do on it. In many cases, data scientists are presented with datasets and tables that do not contain semantic labels, but only the column values. Given the large amount of data they handle, it can be a tedious process to go through every column in a dataset and manually label its semantic type. Thus, we would like to create a method for a machine to determine the semantic type of columns in datasets.

There have been various type detection methods based on regular expression matching or dictionary lookup, but they are dependent on the data being clean and standardized. Sherlock was the first state-of-the-art deep learning method for semantic data type detection, although it has various shortfalls. Sato [1] is an extension of Sherlock [2] that also considers the context of a column within the entire table. With the continuous development of natural language processing, currently, pre-trained large language models have further improved model performance by expanding parameter scale, data scale, and training complexity, and have demonstrated unprecedented language understanding capabilities in various fields. So we aim to examine how well pre-trained large language models can assign semantic labels to columns.

Our method (solution S) is implemented as a program P, which takes a relational table T in CSV format as input, and outputs a list of predicted column names, one for each column in T. The specific steps of our solution are as follows:

1. Input a relational table in CSV format
2. Convert original column names into canonical strings (to use as gold data)
3. Clean table data (e.g. remove null values and table headers if any)
4. Split the table into columns
5. Enter each column (or all values in the table) into a string and fill it into the pre-designed prompt
6. Input prompt into Llama
7. Parse the model output and extract the corresponding column names
8. Output a list of predicted column names

2) Data

The Viznet WebTables corpus is a dataset of tables in which all columns contain 78 valid semantic labels [3]. This dataset is used in other column type detection methods, such as Sherlock and Sato. For our studies, we evaluate the performance of the proposed solutions on the canonical form of the dataset, where all column headers are converted to canonical form to preserve headers with the same meaning but with subtle differences in capitalization and formatting. And we focused only on tables with multiple columns, and we took 500 such tables from the dataset.

Gold data is also necessary in our studies. It not only helps us evaluate the predictions of our methods but also makes the model fine-tuning more accurate. Each table in Viznet WebTables contains the attribute name for each of its columns, so we constructed the required gold data by reading the CSV files in the dataset, extracting and transforming the table headers into canonical form, and using the canonical forms as the ground truth column names.

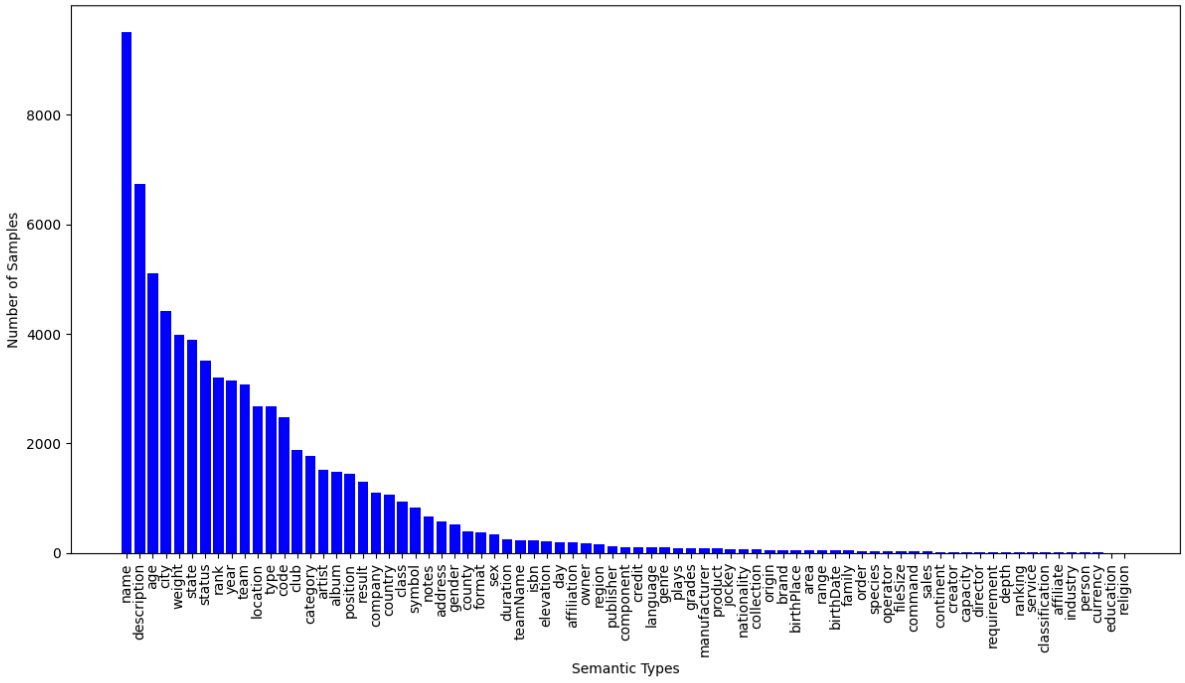

Figure 1: Semantic Type Statistics

3) Performance Measures

Accuracy, Precision, Recall and F_1 score are four metrics that we use to measure the performance of each column semantic type prediction method. Since the semantic type distribution is not uniform, we report two types of average performances using the macro average F_1 score and support-weighted F_1 score. The macro average F_1 score is the unweighted average of the per-type F_1 scores, treating all types equally. The support-weighted F_1 score is the average of per-type F_1 values weighted by support, which is the sample size in the test set for the respective type, and is

therefore more robust to types with small sample sizes compared to macro average F_1 score. The support-weighted F_1 score reflects the overall performance better.

Metrics	Macro Average	Support-Weighted	Meaning
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	$\frac{TP+TN}{TP+TN+FP+FN}$	Proportion of correctly predicted samples to total samples
Precision	$\frac{1}{N}\sum_{i \in types}\frac{TP_i}{TP_i+FP_i}$	$\sum_{i \in types}\frac{n_i}{N}\times\frac{TP_i}{TP_i+FP_i}$	For each type of sample predicted by the model, the proportion of samples that are actually positive class
Recall	$\frac{1}{N}\sum_{i \in types}\frac{TP_i}{TP_i+FN_i}$	$\sum_{i \in types}\frac{n_i}{N}\times\frac{TP_i}{TP_i+FN_i}$	For each type of actual samples, the proportion of correct predictions by the model
F_1	$2\times\frac{precision\times recall}{precision+recall}$	$2\times\frac{precision\times recall}{precision+recall}$	F_1 is the harmonic mean of precision and recall, which reflects the overall performance of the model

Table 1: Metrics Calculation Table

Aside from the above overall evaluation metrics of prediction, we also recorded type-specific metrics for each semantic type. Among them, precision, recall, and F_1-score represent the precision rate, recall rate, and corresponding F_1 value of each type; support represents the actual number of samples of that type in prediction; mis_from_top5 represents the top 5 types among the false positives and what they actually are; and mis_to_top5 represents the top 5 types among the false negatives and what the model thought they were.

type	precision	recall	F_1-score	support	mis_from_top5	mis_to_top5
name	0.8394	0.7278	0.7796	158	[(company’, 6), (description, 5), (category, 3), (artist, 3), (rank, 3)]	[('company', 8), ('description', 7), ('title', 6), ('artist', 6), ('song', 3)]
description	0.4881	0.5616	0.5222	73	[(code, 11), (type, 7), (name, 7), (location, 3), (notes, 2)]	[('year', 8), ('name', 5), ('owner', 3), ('category', 2), ('notes', 2)]

age	0.8481	0.8171	0.8322	82	[(weight, 9), (rank, 3)]	[('weight', 11), ('rank', 2), ('duration', 1), ('location', 1)]
…….						

Table 2: Example of Report's Format and Content

4) Challenges

4.1) Hardware

The CSL machine instgpu-05 has eight NVidia A40 48GB GPUs and 1 TB RAM. And the CSL machine instgpu-01/2/3/4 has eight NVidia Founders Edition 2080TI GPUs, each with 11GB of GPU memory. These hardware resources are shared, which means that we need to coordinate the machine usage time with others. At the same time, the disk and memory of these hardware also limit the size of the model that can be inferred or trained, as each user has only a 100GB disk quota. After reading details about Llama [4], we realized that we can store Llama2-7B, Llama2-13B, and Llama3-8B on our disk, and Llama2-70B and Llama3-70B come in quantized versions that can also be stored on our disk [13, 14], and we can delete previous models when we need to try new ones.

4.2) Model Selection

Finding a pre-trained LLM that is compatible with our limited computing resources and as powerful as expected is another related challenge. One option is to call some commercial interfaces of powerful pre-trained models such as GPT-3.5, but that is paid and cannot be optimized for the code. Another option is open-source models such as Llama2 and Llama3, but we need to ensure that they can run on our limited machine resources and complete the experiment within an effective time. After reading details about Llama [4], we realized that Llama2-7B, Llama2-13B, and Llama3-8B can run on the instgpu-05 machine, and Llama2-70B and Llama3-70B come in quantized versions that can run on the CSL machines [13, 14].

4.3) Prompt Engineering

When using LLMs, the prompt must be written in a certain way to get an output close to the desired output. In our study, we wanted to output either a single prediction or a list of predictions, but LLMs usually output paragraphs of natural language. After reading about Llama prompt engineering, we learned that the best way to get a standardized output is to request JSON format, which supports lists [8]. Thus, we decided to tell Llama to only output valid JSON. The Llama2-7B and Llama2-13B models struggled with confining their outputs to valid JSON, but Llama2-70B and the Llama3 models were mostly able to output only JSON.

4.4) Output Parsing

A related problem is parsing the output when it is not in JSON format. For the single-column methods on Llama2-7B and Llama2-13B (discussed in section 5.1), we noticed that the column prediction usually appeared before the first period in the paragraph, so we extracted the last word before the period and used that as the prediction. If there was no period, we would extract the last word in the whole paragraph. For the multi-column methods (also discussed in section 5.1), we instead returned blank predictions for all the columns whenever the output was not in JSON format.

A similar problem is that in some cases, for the multi-column methods, Llama would give a different number of predictions than the actual number of columns. Whenever this happened, we also returned blank predictions for all the columns.

4.5) Fine Tuning

Fine-tuning technology further trains a model that has been pre-trained on a large dataset on a task-specific dataset, slightly adjusting the model parameters, to optimize the model's performance on a specific task [10]. Fine-tuning can help us further improve the performance of the model, but there are also the following difficulties.

First, from a data perspective, the quality of the dataset has a great impact on the performance of the model. If the distribution of the specific task dataset used in fine-tuning is different from the pre-training data's distribution, this mismatch may lead to model performance degradation. So before fine-tuning, we clean and transform the original data into gold data to build a dataset suitable for fine-tuning tasks.

Second, from a training perspective, training large models usually requires a large amount of computing and storage resources, including high-performance GPUs, a large amount of physical memory and GPU memory. These resources are limited in this project and will limit optimization results. In addition, the training of large models is usually time-consuming, and the impact of fine-tuning on project progress needs to be constantly evaluated. For this training problem, we can use some advanced fine-tuning techniques such as parameter-efficient fine-tuning to reduce the required training parameters and computing resources, so that it can take less time to complete the training task.

5) Solutions

5.1) Methods

LLaMA comes in various forms. When we started our study, Llama2 was the newest set of models, and they came in versions with 7B parameters, 13B parameters, and 70B parameters. On April 18th, 2024, Llama3 was released in 8B parameter and 70B parameter versions.

We made a pre-specified prompt template to enter into Llama [6, 7]. First, we tell Llama to produce results in JSON format that should not include any harmful, unethical, racist, sexist, toxic, dangerous, or illegal content. Then we provide the list of 78 valid column types and state that the predictions should only be among these types. Then we input either the full table or just

one column of each table; this will be discussed in the next paragraph. Due to the limit of the input size for Llama, we use only the first 20 rows of each table. Finally, we tell Llama to make one prediction if we input only one column, or make N number of predictions if we input the full table with N columns.

There are two main ways a table can be processed to get semantic column labels. One is to enter each column into a separate prompt and get only the prediction for that column, and we call this the single-column method. First, the CSV file is loaded into the program as a Pandas DataFrame. Then we iterate through each column and enter the column values into our pre-specified prompt template, where the column values are separated by a comma and space. Inside the prompt, each column looks something like this: "value1, value2, …". After doing this for all columns, this creates a list of prompts for the LLM to process. We then enter the prompts into the LLM and get predictions for the column names. The other way is to enter the whole table into one prompt and get predictions for all the columns, and we call this the multi-column method. First, we load the entire CSV as a string, delete the column names, and then paste the remaining text into our pre-specified prompt template without any modification. In our prompt, we specify that the model should output a JSON file with a list.

We also had to figure out how to extract the type predictions from Llama's output. Llama is designed for natural language conversation, so it usually outputs a paragraph in natural language, but we learned that it can be instructed to output JSON [7, 8]. Thus, we tried to instruct Llama to output the predictions as a JSON list. Some of the later models, such as Llama2-70B and both Llama3 models, generally followed the JSON instruction, so we used a Python function to convert the JSON into a dictionary and return the list, and if the JSON was invalid, we would return blank predictions. On the other hand, the outputs of Llama2-7B and Llama2-13B often contained invalid JSON format or additional text outside of the JSON. Thus, we took a different approach, one for single-column and one for multi-column. For the single-column method, we noticed that the predicted column name would often appear before the first period in the paragraph (if there was one), so we split the paragraph by periods and took the last word before the first period as the prediction. For the multi-column method, we removed the text before and after the JSON, and then ran the JSON through the Python function, and if the JSON was invalid, we would return blank predictions.

In our first solution, which is our baseline, we use the single column method to process the table and use Llama2-7B as the underlying model. And we get different solutions by replacing different parts of the pipeline with different methods, such as replacing single column with multi column method or replacing Llama2-7B with more advanced models. We show the whole pipeline in Figure 2.

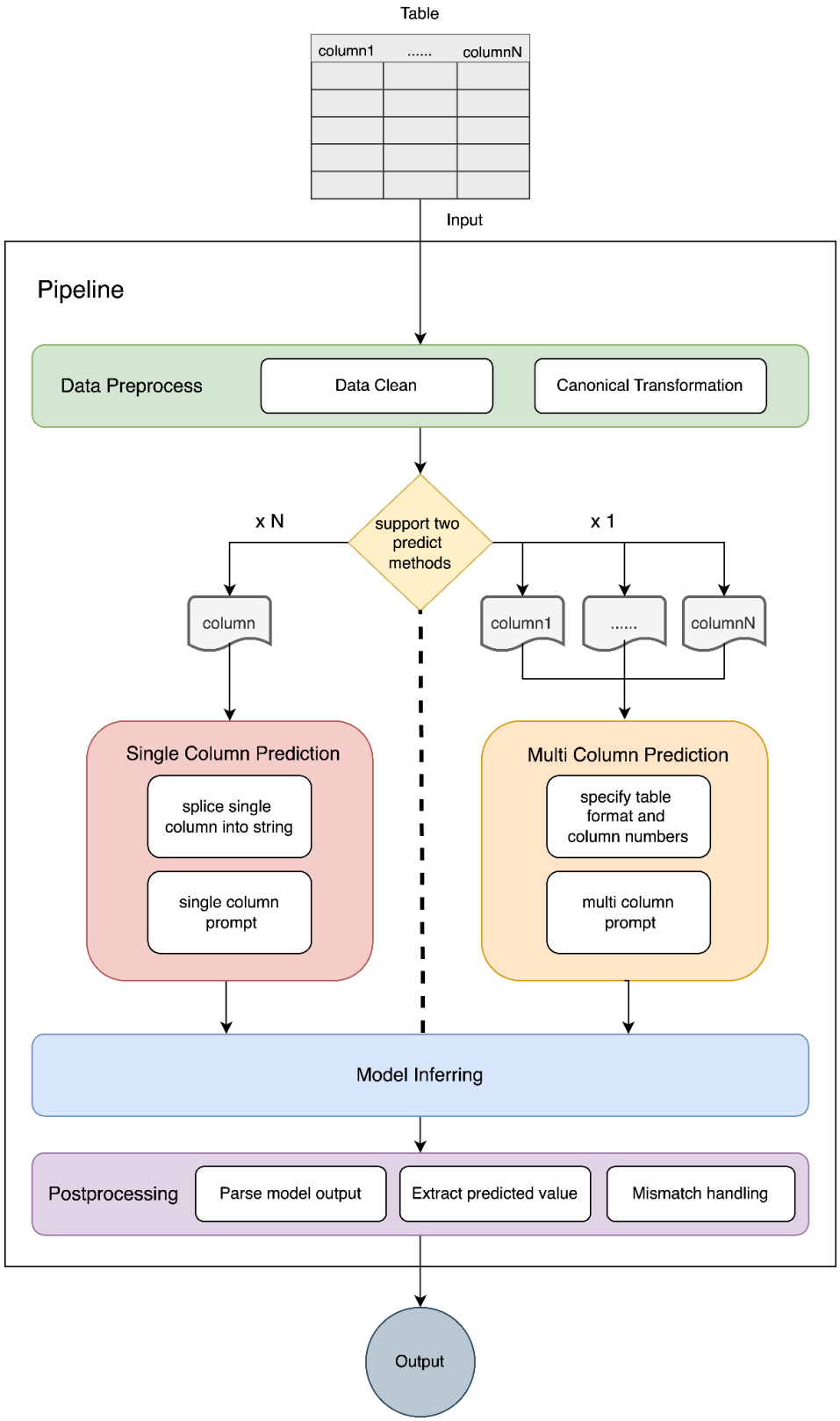

Figure 2: Solution Pipeline

5.2) Experiments (Pipelines)

The table below shows performance benchmarks for the chat-tuned versions of Llama2 and Llama3. We will refer to this table when discussing our experiments.

Benchmark	**Llama 3 8B**	**Llama 2 7B**	**Llama 2 13B**	**Llama 3 70B**	**Llama 2 70B**
MMLU (5-shot)	68.4	34.1	47.8	82.0	52.9
GPQA (0-shot)	34.2	21.7	22.3	39.5	21.0
HumanEval (0-shot)	62.2	7.9	14.0	81.7	25.6
GSM-8K (8-shot, CoT)	79.6	25.7	77.4	93.0	57.5
MATH (4-shot, CoT)	30.0	3.8	6.7	50.4	11.6

Table 3: Benchmarks for Instruction Tuned Models [4]

Baseline (Solution 1)*: *Llama2-7B-Single
Our first pipeline is the simplest pipeline, and we use that as our baseline. In this pipeline, the prediction method module is single-column prediction, the model is Llama2-7B, and the last word of the generated output is directly extracted in the postprocessing part.

The experiment's results (given in Table 3) show that the accuracy, precision and recall of this solution all show a low level, which is considered to be due to the insufficient capabilities of the Llama2-7B model itself. Therefore, our next step is to first use the fine-tuning method to optimize the model to improve its performance on our column profiling task.

Solution 2: Llama2-7B-Single-fine-tuned
In order to verify the hypothesis of the previous solution, that is, the model itself caused the low results, we used the idea of ablation experiments and only optimized the model without changing other steps (data preprocessing, single column method, output extracting method). To construct the dataset for fine-tuning, we split the original dataset and used each column (gold data) as a training sample, with the column values as features and the column header as the label. During the fine-tuning process, we used Meta's official open-source fine-tuning tool llama-recipes (https://github.com/meta-llama/llama-recipes), which is a script for fine-tuning Meta Llama series models with composable FSDP & PEFT methods to cover single/multi-node GPUs. The

fine-tuning output is an adapter model, which can be added to the original model to improve performance[9, 10, 11].

Our dataset used for fine-tuning and the fine-tuned model have been published on the huggingface platform (https://huggingface.co/sadpineapple) and can be directly imported and used with the interface of the transformers package in Python[10].

Through the data comparison (given in Table 3) after the experiment, it is found that almost all the metrics after fine-tuning are higher than the previous results, which proves that the ability of the model does significantly affect the prediction results of column semantics. But unfortunately, due to the physical machine limitations, we cannot train larger models or more epochs on existing equipment. At the same time, we also found many strange mismatches from the result report, such as predicting "name" as "any", or the output predicted name outside of the specified 78 types. This is considered to be because Llama2-7B can not understand the meaning of prompts well and can not generate results in JSON or other formats, and also we don't have corresponding post-process logic. To sum up, therefore we decided to use other models such as Llama2-13B, and explored prompt engineering to optimize the results.

Solution 3: Llama2-13B-Single

Our next pipeline involved a single-column method using Llama2-13B, which requires two NVidia A40 GPUs. Aside from the pretrained model underlying the code, this method is the same as the previous single-column methods. Overall, it returned higher accuracy than Llama2-7B-Single and Llama2-7B-Single-fine-tuned, with a weighted average F_1 score of 0.4949. This is consistent with the metrics in Table 3; Llama2-13B outperforms Llama2-7B in all benchmarks. But the invalid JSON problem still exists, so we still use the method of parsing the last word before the first period.

Solution 4: Llama2-70B-Single

We wanted to see if a model with more parameters would return better performance. Thus, we decided to replace Llama2-13B with a quantized version of Llama2-70B, since we did not have enough disk space to store the regular 70B model, and this model requires eight GPUs. The performance was similar to Llama2-13B-Single, with a weighted average F_1 score of 0.4765. This is consistent with the metrics in Table 3; Llama2-70B outperforms Llama2-13B in certain benchmarks but underperforms in others. But the invalid JSON problem still exists, so we still use the method of parsing the last word before the first period.

Solution 5: Llama2-70B-Multi

One potential advantage of using a multi-column method over single-column methods is that it provides more context for the column prediction. For example, a table may have the column names [name, birthPlace, birthDate]. If we look at the birthPlace column, we would get a list of cities. If we have all columns, it is easier to infer that this list of cities refers to the birthplace of the person whose name is listed in that row. However, if we only see the list of cities, it would make sense to give it the more general prediction of "location" or "city" [1]. In our studies, we wanted to see whether introducing context helps Llama make more accurate predictions.

Unfortunately, aside from the macro-average metrics for Llama2-7B-Single, the performance was worse than any of the previous single-column methods, with a weighted average F_1 score of 0.2784. In many cases, the method returns a different number of predictions than the actual number of columns. We hypothesize that this is because Llama2 has difficulty understanding the structure of the table and how to separate the columns. This problem is further discussed in section 6.1. Based on this hypothesis, we also hypothesize that if Llama2 has difficulty separating the columns, it gets confused about which values belong to which columns, which would confuse the model when making predictions. Additionally, the invalid JSON problem still occurs, and unlike the single-column methods where we can extract words in the paragraph, we make blank predictions whenever this problem occurs, which could further decrease performance.

Solution 6: Llama3-8B-Single

On April 18th, 2024, Meta released Llama3 in 8B and 70B forms. Based on the benchmarks in Table 3, it seems like even Llama3's 8B model outperforms all of the Llama2 models. Thus, we wanted to see whether Llama3 would perform better than Llama2 at our task.

To start out, we used the Llama3-8B model, which requires one NVidia A40 GPU. With a weighted F_1 score of 0.5996, this method gave significantly higher accuracy metrics than any of the Llama2 methods. This is consistent with Llama3-8B outperforming all Llama2 models in Table 3 benchmarks. Additionally, it followed the instructions about only outputting valid JSON, so we changed our postprocessing method to convert the JSON output into a dictionary and extract the prediction from it. This suggests that Llama3 is also better at following user instructions compared to Llama2.

Solution 7: Llama3-8B-Multi

We wanted to see if Llama3-8B would be better at using context compared to Llama2-70B-Multi. With a weighted average F_1 score of 0.2784, performance was slightly worse than Llama3-8B-Single, but still better than Llama2-70B-Multi. The number mismatch error is less prevalent in this model compared to Llama2-70B-Multi, but it still occurs. This is consistent with Llama3-8B outperforming Llama2-70B in all benchmarks, which suggests that Llama3 could be better at understanding table structure compared to Llama2, but it still has considerable difficulty.

Solution 8: Llama3-70B-Single

We wanted to see if Llama3-70B would return better performance than the 8B model. With a weighted average F_1 score of 0.6186, this model gave slightly better performance than the Llama3-8B-Single. This is consistent with the metrics in Table 3; Llama3-70B outperforms Llama3-8B in all benchmarks.

Solution 9: Llama3-70B-Multi

With a weighted average F_1 score of 0.6585, this was the first multi-column method that seemed to improve over its single-column counterpart. Only weighted average precision was slightly lower than for Llama3-70B-Single; all other metrics were higher. In our run of this method, the number mismatch error never occurred. This suggests that Llama3-70B is much better than Llama2 or Llama3-8B at understanding table structure.

5.3) Metrics

Method	Macro Average			Support-Weighted Average			Accuracy
	Precision	**Recall**	**F_1**	**Precision**	**Recall**	**F_1**	
Llama2-7B-Single	0.0732	0.0496	0.0524	0.4952	0.3103	0.3203	0.3103
Llama2-7B-Single-fine-tuned	0.1225	0.0914	0.0908	0.4726	0.3648	0.3553	0.3648
Llama2-13B-Single	0.1317	0.0883	0.0944	0.6360	0.4841	0.4949	0.4841
Llama2-70B-Single	0.1716	0.1299	0.1329	0.6005	0.4882	0.4765	0.4882
Llama2-70B-Multi	0.1242	0.0690	0.0789	0.4835	0.2689	0.2784	0.2689
Llama3-8B-Single	0.2449	0.1949	0.1981	0.6803	0.5947	0.5996	0.5947
Llama3-8B-Multi	0.1742	0.1383	0.1416	0.4646	0.3878	0.3817	0.3878
Llama3-70B-Single	0.3399	0.3232	0.2973	0.7234	0.6147	0.6186	0.6147
Llama3-70B-Multi	0.3653	0.3290	0.3233	0.7061	0.6572	0.6585	0.6572

Table 4: Performance metrics for our methods

6) Misc

6.1) Limitations

One limitation of our method is that it sometimes returns column names outside the 78 types. In many cases, it uses a value in the column as the predicted column name, and this often happens when the same value appears frequently in a column. We have several hypotheses on why this is the case. One is that Llama has difficulty understanding the instruction part of the prompt, either because of its structure or because it doesn't know how to confine its output to the 78 types. Another is that Llama was designed for natural language processing and output, so its decisions are based on intuition rather than instructions. Thus, while in some cases the predicted type would be logically correct, it does not conform to the 78 types. To mitigate this issue, one approach is to improve prompt engineering to better assist Llama in understanding the task at hand. Another approach is to fine-tune the Llama models for this task.

Another limitation is that for the multi-column methods, it sometimes returns a different number of predictions than there are columns. This problem didn't appear in our runs of Llama3-70B-Multi, but it is still something to consider, as it appeared in the other models. We hypothesize that this is because Llama has a hard time determining how to read the CSV and

how to separate the columns. To solve this problem, one approach is to improve the prompt engineering to better assist Llama in understanding the structure of the table.

6.2) Proposed future work

One next step is to further optimize the fine-tuning process to improve model performance. Due to resource constraints, currently, we were only able to fine-tune Llama2-7B, and that still returned worse performance than some of the other models. We consider fine-tuning the Llama3 model when more resources are available and consider using some advanced fine-tuning techniques such as parameter-efficient fine-tuning to further reduce the required training parameters and computing resources. Additionally, we also plan to add more information (such as column statistics, column neighbors, table global information, etc.) to the fine-tuning dataset to explore and improve the dataset for the column profiling task.

Another step is to try more prompt engineering. One thing we can do is add richer contextual information to the prompts to help the model understand task requirements and background knowledge better. For example, we can add some database terms to the prompts to help the pre-trained model better understand the structure of the table and the relationship between schemas and data instances. In addition, we can also try different types of prompts for experiments. The simplest way is to manually explore the most suitable prompt content, such as replacing direct prompts with indirect prompts, switching different language styles, etc. Another possible approach is to use genetic algorithms, reinforcement learning, or other search techniques to automatically generate, test, and find the most effective prompts.

Another step would be to try a different LLM. We used Llama because it is open-source and can run on our resources, but trying this task on various LLMs would give us more insight on how one can use LLMs for the column profiling task. A related step would be to try this method on every future release of Llama, the way we tried it on Llama3 when that set of models was released.

A more radical approach would be to combine Llama with an existing state-of-the-art column profiling method, such as Sato. The authors of Sato state that “A promising avenue of future research is to combine our multi-column model with BERT-like pre-trained learned representation models” [1]. Sato was released in 2019, which is before Llama was released in 2023, so at the time they used BERT, a LLM released by Google in 2018 [12], as a reference point. Combining Llama and Sato could leverage the benefits of both methods.

7) Conclusions

In our study, we tried different sizes of pretrained LLMs and used different table processing methods and prompts to explore LLMs’ ability in column profiling. Our findings suggest that although LLMs can technically perform the task of semantic column type detection, more work needs to be done for them to do the task well. Our overall best model achieved a weighted average F_1 score of 0.6585. In contrast, Sherlock achieves a weighted average F_1 score of 0.879 on multi-column tables, and Sato improves on Sherlock’s performance further with a weighted average F_1 score of 0.925 [1]. Sherlock’s and Sato’s F_1 scores are based on more data than what

we used, but even if the scores are not directly comparable, the considerable difference between them implies that there are many improvements needed before Llama3 can be used for semantic column type detection. Because Sato and Sherlock were designed specifically for this task, they are structurally incapable of providing invalid predictions the way Llama does, which likely contributes to their better performance. Similarly, they were trained directly on the original tables with gold data, whereas pretrained Llama is trained for a different set of tasks. Thus, as mentioned in section 6.2, fine-tuning Llama for the column profiling task is one approach to improve performance.

8) References

[1] Zhang, Dan, et al. "Sato: Contextual semantic type detection in tables." *arXiv preprint arXiv:1911.06311 (2019).*

[2] Hulsebos, Madelon, et al. "Sherlock: A deep learning approach to semantic data type detection." *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019.

[3] Hu, Kevin, et al. "VizNet: Towards a large-scale visualization learning and benchmarking repository." *Proceedings of the 2019 CHI conference on human factors in computing systems*, 2019.

[4] "Llama Model Card." *GitHub*, Meta, 2024, https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

[5] Touvron, Hugo, et al. "Llama 2: Open foundation and fine-tuned chat models." *arXiv preprint arXiv:2307.09288 (2023)*.

[6] Holtz, Charlie. "A guide to prompting Llama 2." *Replicate*, 14 August 2023, https://replicate.com/blog/how-to-prompt-llama#system-prompts.

[7] Boraks, Eliran. "Llama 2 Prompt Engineering — Extracting Information From Articles Examples." *Medium*, 14 November 2023, https://medium.com/@eboraks/llama-2-prompt-engineering-extracting-information-from-articles-examples-45158ff9bd23.

[8] “26 prompting tricks to improve LLMs.” *SuperAnnotate*, 9 January 2024, https://www.superannotate.com/blog/llm-prompting-tricks.

[9] Schmid, Philipp. “Extended Guide: Instruction-tune Llama 2.” *Philschmid*, 26 July 2023, https://www.philschmid.de/instruction-tune-llama-2.

[10] “Fine-tune a pretrained model.” *Hugging Face,* https://huggingface.co/docs/transformers/training. Accessed May 9, 2024.

[11] Awan, Abid Ali. “Fine-Tuning LLaMA 2: A Step-by-Step Guide to Customizing the Large Language Model.” *DataCamp*, 2023, https://www.datacamp.com/tutorial/fine-tuning-llama-2.

[12] Devlin, Jacob and Chang, Ming-Wei. “Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing.” Google, 2 November 2018, https://research.google/blog/open-sourcing-bert-state-of-the-art-pre-training-for-natural-language-processing/.

[13] Jobbins, Tom. “Llama2-70B-GPTQ.” *HuggingFace*, 2023, https://huggingface.co/TheBloke/Llama-2-70B-GPTQ.

[14] Jiang, Hao. “Llama3-70B-Instruct-GPTQ.” *HuggingFace*, 2024, https://huggingface.co/TechxGenus/Meta-Llama-3-70B-Instruct-GPTQ.