Efficient Table Generation for Zero-Shot Column Type Annotation

Ehsan Hoseinzade [1] **Ke Wang** [1]

Abstract

This study addresses the challenge of automatically detecting semantic column types in relational tables, a key task in many real-world applications. Zero-shot modeling eliminates the need for labeled tabular data, making it ideal for scenarios where data collection is costly or restricted due to issues such as privacy concerns. However, existing zero-shot models often perform poorly when dealing with a large number of semantic types and show limited understanding of tabular structure. We propose an efficient zero-shot table generation approach that constructs structured pseudo-tables using publicly available data. Fine-tuning an open-source LLM on these synthetic tables enables it to better capture tabular structure and improve column type annotation. Experiments show that our method outperforms state-of-the-art zero-shot and few-shot models by at least 10.4% and 7%, respectively.

1. Introduction

Column type annotation, i.e., identifying or tagging the semantic types of columns, or classes, inside a table, is crucial for different information retrieval tasks like data integration (Hai et al., 2023), data cleaning (Limaye et al., 2010; Kandel et al., 2011), schema matching (Rahm & Bernstein, 2001), and data discovery (Fernandez et al., 2018b;a). One emerging application, for example, is automatically tagging sensitive columns in a table, such as personal information, before deciding what information can be released.

Supervised learning-based methods (Hulsebos et al., 2019; Zhang et al., 2019; Deng et al., 2022; Suhara et al., 2022; Sun et al., 2023; Hoseinzade & Wang, 2024) have shown promising results. These models primarily leverage BERT's pre-training on large-scale textual corpora, fine-tuning it for labeled tabular training data by adding task-specific output layers to classify each column into a predefined set of semantic types. However, these models are highly dependent on labeled tabular training data. Collecting high quality labeled tabular training data is a resource-intensive and time-consuming process. In many cases, such data might not exist in the required format or it could be confidential because privacy concerns and regulations such as HIPAA and GDPR impose restrictions on sharing sensitive data in domains like healthcare, finance, and government, making it nearly impossible to collect and use training data in these fields.

Large Language Models (LLMs) have been proposed as a solution to data availability challenges to perform zero-shot column type annotation without the need for labeled tabular datasets (Kayali et al., 2024; Korini & Bizer, 2023; Feuer et al., 2024). These models, pre-trained on extensive textual corpora are attractive for scenarios where labeled data is difficult to obtain. However, despite their potential, current LLM-based zero-shot models for column type annotation suffer from the following limitations:

Performance: LLM performance in zero-shot classification often falls short, particularly when dealing with closely related classes like "addressRegion," "addressLocality," "streetAddress," and "PostalAddress" (Feuer et al., 2024). These models struggle to distinguish such classes without learning subtle differences between them, unlike supervised models that rely on labeled tabular data for this purpose.

Structure: LLMs, being primarily trained on unstructured textual data, struggle to learn the structural relationships between columns within tables (Li et al., 2024; Sui et al., 2024). So, zero-shot models based on LLMs are less effective at capturing table-specific details like values in the same column or rows, unlike supervised models, which directly learn them from tabular training data.

The question is how to design a zero-shot learning model that can understand different classes and table structures without the need for user-provided tabular training data. We propose a novel zero-shot framework, **ZTab**, that leverages two LLMs to meet these requirements. ZTab uses a description LLM to generate representative sample values, called class descriptions, for each semantic type (e.g., Country: Canada, UK, France), and use class descriptions to construct *pseudo-tables* labeled by column semantic types. Unlike prior zero-shot methods that rely solely on inference-

[1]Simon Fraser University, Burnaby, Canada. Correspondence to: Ehsan Hoseinzade <ehoseinz@sfu.ca>.

Proceedings of the 1st ICML Workshop on Foundation Models for Structured Data, Vancouver, Canada. 2025.

time prompting, ZTab fine-tunes an annotation LLM using these pseudo-tables to learn differences of classes. ZTab addresses the above challenges as follows. For **performance**, like supervised learning, the fine-tuning on the pseudo-tables helps better distinguish between similar semantic types, and like zero-shot learning, no user-provided training data is needed. For **structure**, the fine-tuning on pseudo-tables with realistic column combinations and value patterns allows the model to learn both relationships between columns and typical distributions within columns.

Our key contributions are as follows:

- **Efficient and diverse fine-tuning table generation**: Instead of directly generating one table at a time from LLMs, which often produces low-quality tables (Berkovitch et al., 2024), ZTab generates class descriptions using an LLM and constructs fine-tuning pseudo-tables using class descriptions. This approach allows efficient generation of a large number of diverse tables needed for fine-tuning and robust generalization of the annotation LLM.
- **Improved zero-shot performance**: ZTab's fine-tuning on an LLM using pseudo-tables improves the existing zero-shot performance by at least 10.4% on average over multiple datasets (Table 1). The performance is measured by micro-F1 score.

2. Problem Definition

We study *data generation-based zero-shot column type annotation*, where the goal is to assign semantic types $(c_1, c_2, \ldots, c_n)$ to the columns of a table $T = (t_1, t_2, \ldots, t_n)$, with each c_i from a predefined set C, without using labeled training data. The solution has two phases: a learning phase that builds a model from schema-level supervision, and a deployment phase that applies it to annotate given tables.

Learning Phase: The inputs are: (1) a predefined set of semantic types $C = \{c_1, c_2, \ldots, c_m\}$ (e.g., *Name*, *Date*), and (2) a collection of table schemas $S = \{S_1, S_2, \ldots, S_k\}$, where each schema $S_i = \{h_1, h_2, \ldots\}$ is a list of column headers selected from C. For example, $S_i = \{$"Country", "Locality"$\}$ represents a table with two columns labeled as "Country" and "Locality". The learning phase uses only these schema-level inputs to build a model M_a, without access to any actual table content.

Deployment Phase: Given a new table T without headers and candidate type set C, model M_a (from learning phase) predicts a semantic type from C for each column in T.

3. ZTab

ZTab operates in two phases. In the **Learning Phase**, it fine-tunes an open-source LLM M_a (the annotation LLM) using pseudo-tables constructed from table schemas in S and example values for semantic types in C, called *class descriptions*. These class descriptions are generated by another LLM M_d (the description LLM), pretrained on a broad corpus. Instead of generating full tables, ZTab uses M_d only to produce representative values for each type, addressing the lack of labeled data. Pseudo-tables are then formed by populating the schemas in S with these values. This use of synthetic yet structured pseudo-tables during fine-tuning enables M_a to learn type annotation in tabular contexts and generalize to unseen tables in a zero-shot setting. In the **Prediction Phase**, the fine-tuned M_a predicts semantic types for columns in given tables.

Algorithm 1 Learning Phase of ZTab

Require: Set of classes C, Table schema collection S, Annotation LLM M_a, Description LLM M_d, Schema sampling ratio r, Class description size e, row size k
Ensure: Fine-tuned LLM M_a

{*step 1: class descriptions generation*}
$D \leftarrow$ empty list
for each class c_i in C **do**
  $d \leftarrow$ **ClassDescriptionGeneration**(c_i, e, M_d)
  Add d to D
end for
{*Step 2: handling missing classes*}
$C_{missing} \leftarrow C \setminus \{\bigcup_{S_i \in S} S_i\}$
$S_{manual} \leftarrow \{\{c_i\} : c_i \in C_{missing}\}$
$S \leftarrow S \cup S_{manual}$
{*Step 3: fine-tuning*}
for each epoch **do**
  $S_{rand} \leftarrow$ randomly select r percent of S
  $Prompts, Labels \leftarrow \emptyset$
  for each $S_i = \{h_1, ..., h_n\}$ in S_{rand} **do**
    $Table_i(t_1, ..., t_n) \leftarrow$ **TablePopulation**(S_i, D, k)
    $prompt_i \leftarrow$**PromptConstruction**($Table_i$, C)
    Add $prompt_i$ to $Prompts$
    Add $label_i = (h_1, ..., h_n)$ to $Labels$
  end for
  for each batch $(Prompts_{\text{batch}}, Labels_{\text{batch}})$ **do**
    $Outputs \leftarrow M_a(Prompts_{\text{batch}})$
    $Loss \leftarrow Loss(Outputs, Labels_{\text{batch}})$
    $M_a \leftarrow UpdateWeights(M_a, Loss)$
  end for
end for
return M_a

Let us detail these phases.

3.1. Learning

The learning phase, described in Algorithm 1, consists of three main steps described below:

Class description generation: For each semantic type $c_i \in C$, we query an LLM M_d, the description LLM, to generate up to e examples of c_i serving as its class description. The detail is captured by the function **ClassDescriptionGeneration**(c_i, e, M_d) explained shortly. These class

descriptions, denoted by D, are used to generate fine-tuning pseudo-tables during the fine-tuning step.

Handling missing classes: This step creates one table schema for each class in C not contained in any table schema $S_i \in S$ and add these schemas to S. The intention is to ensure that all semantic types/classes in C are represented in the fine-tuning process.

Fine-tuning: This step fine-tunes the annotation LLM M_a in multiple epochs. During each epoch, it randomly selects r percent of the table schemas from S, denoted S_{rand}, and constructs a pseudo-table $Table_i$ of k rows for each schema S_i in S_{rand}, done by the function **TablePopulation**(S_i, D, k). It then represents $Table_i$ by creating one prompt for each column in the table, done by the function **PromptConstruction**($Table_i, C$), and stores the prompts in $prompt_i$. The prompts $Prompt$ and the table headers $Labels$ for the schemas in S_{rand} are used to fine-tune M_a (i.e., lines 18-22), on a batch basis as in (Brown et al., 2020). The fine-tuning cost is determined by the schema sampling ratio r and the row size k. A small value of k is often sufficient for good performance of ZTab. More details are given in appendix A.

Below, we explain the bold face functions in Algorithm 1.

ClassDescriptionGeneration(c_i, e, M_d): For the class c_i, the description LLM M_d is provided with the prompt *"Generate e real-world examples of the semantic type c_i commonly found in web tables."*, where e is the class description size. In response, M_d generates up to e instances for the class c_i. For example, with c_i being the class 'City' and $e = 50$, M_d will generate up to 50 city names to form the class description for the class City. Note that M_d can be replaced with any knowledge base such as DBpedia or Wikidata; we choose an LLM pre-trained on the world corpus for better generalization of models.

TablePopulation(S_i, D, k): This function populates the table schema S_i by randomly selecting k values from the corresponding class description in D for each semantic type in S_i. Thanks to the random selection of values from the class description, for the same schema S_i selected in different epochs, the table generated for S_i could be very different, allowing the fine-tuning to encounter a diverse range of training examples, which is essential for a better model generalization. An alternative is to generate tables (one table at a time) using LLMs directly, but it tends to produce low-quality tables as discussed in (Berkovitch et al., 2024). Our fine-tuning benefits from a large number of diverse tables that are generated efficiently from class descriptions.

PromptConstruction ($Table$, C): For a given $Table = (t_1, t_2, \ldots, t_n)$ with n columns and k rows and a collection of semantic types $C = \{c_1, c_2, \ldots, c_m\}$, this function generates n prompts, one for each column in $Table$. Figure 1 shows the prompt generated for a column $t_i \in Table$, which has four parts: (1) **Introduction**: The general instruction about the table structure. (2) **Table Presentation**: The table data presented column-by-column. (3) **Task Description**: The instruction for the LLM to annotate the target column. (4) **Target Column**: The target column for which the model is expected to predict the semantic type. While each prompt focuses on prediction for one target column, the entire table is presented in each prompt to help infer the target column's semantic type in the context of other columns in the table.

These are values of columns in a table. Each column starts with Column: followed by the values of that column. First, look at all the columns to understand the context of the table.

Column 1: $t_{11}, t_{12}, \ldots, t_{1k}$

Column 2: $t_{21}, t_{22}, \ldots, t_{2k}$

...

Column n: $t_{n1}, t_{n2}, \ldots, t_{nk}$

Your task is to annotate the Target Column using one semantic type that matches the values of the Target Column and the context of the table from the following list: $c_1, c_2, \ldots, c_m$.

Target Column: $t_{i1}, t_{i2}, \ldots, t_{ik}$

Semantic Type:

Figure 1. The prompt for the target column t_i in a table with n columns and k rows.

Algorithm 2 Prediction Phase of ZTab

Require: New Table $T = (t_1, t_2, \ldots, t_n)$, set of semantic types/-classes C, Fine-tuned Model M_a from Algorithm 1
Ensure: Predicted Class for Each Column in T
1: $Prompts \leftarrow$**PromptConstruction**(T, C)
2: **for** each $prompt_i$ corresponding to column t_i in $Prompts$ **do**
3: $Output_i \leftarrow M(prompt_i)$ {*label remapping*}
4: $h_i \leftarrow \arg\max_{c_j \in C}$ cosine_similarity$(E(M_a, Output_i), E(M_a, c_j))$
5: **end for**
6: **return** $(h_1, ..., h_n)$

3.2. Prediction

Algorithm 2 presents the prediction phase. In this phase, the fine-tuned model M_a produced by Algorithm 1 is applied to a new table T without column headers to annotate its columns using the semantic types in C. First, **PromptConstruction** (T, C) is used to generate the prompts for the columns in T. $M_a(prompt_i)$ denotes the tokens generated by M_a for $prompt_i$, which may not be a member of C. *label remapping* is used to map the generated tokens to the most similar class in C_{pred}, i.e., h_i, where similarity is measured based on the embeddings $E(M_a, \cdot)$ extracted from M_a. Finally, $(h_1, \cdots, h_n)$ are returned as the predicted semantic types for the columns $(t_1, \cdots, t_n)$.

	Zero-shot						Few-shot		Supervised Learning	
	(Ours)		Baseline				Reference		Reference	
Dataset	ZTab	ZTab$_{w/o\ S}$	ArcheType$_{ZS}$	GPT$_{col\text{-}ZS}$	GPT$_{table\text{-}ZS}$	Chorous	GPT$_{col\text{-}FS}$	GPT$_{table\text{-}FS}$	Doduo	ArcheType$_{FT}$
SOTAB$_{sch}$	**78.0**	72.1	40.1	55.5	64.0	60.3	65.4	68.2	86.3	85.1
SOTAB$_{sch\text{-}s}$	**76.2**	72.1	40.1	55.5	64.0	60.3	65.4	68.2	81.1	83.0
SOTAB$_{dbp}$	**78.5**	74.2	48.3	59.4	70.0	64.8	67.0	72.4	85.2	83.6
T2D	**96.2**	93.9	81.6	87.9	89.4	83.4	90.1	92.1	91.1	88.0
Average	**82.2**	78.1	52.5	64.6	71.8	67.2	72.0	75.2	85.9	84.9

Table 1. Micro-F1 score of different methods.

4. Evaluation Method

Datasets: We use four datasets: T2D (Chen et al., 2019), SOTAB$_{sch}$, SOTAB$_{sch\text{-}s}$, and SOTAB$_{dbp}$ (sot, 2023), summarized in Table 2. For each dataset, the schema collection S is extracted by including the table header (i.e., the list of semantic types for columns in the table) of each training table, with duplicates preserved and C set to class list of dataset.

Dataset	# Class (C)	# Training Tables	# Test Tables
SOTAB$_{sch}$	82	44,769	609
SOTAB$_{sch\text{-}s}$	82	10,631	609
SOTAB$_{dbp}$	46	37,631	279
T2D	37	160	109

Table 2. Summary of datasets.

Metric: We evaluate the performance by the *micro F1-score* collected on test tables, following previous works (Feuer et al., 2024; Korini & Bizer, 2023; 2024). All the F1-scores are multiplied by 100 (e.g., 80% is written as 80).

ZTab: Our main model, **ZTab**, uses ChatGPT-3.5 as the description model (M_d) and Qwen-7B (Bai et al., 2023) as the annotation model (M_a). Qwen-7B is fine-tuned for 20 epochs using LoRA (rank 256), with batch size 1, gradient accumulation 8, and learning rate 1×10^{-5}. We set the class description size to $e = 500$, row size to $k = 3$, and schema sampling rate r to 2.5% for SOTAB datasets and 100% for T2D. Further analysis is provided in Appendix A.

ZTab$_{\text{w/o } S}$: This variant removes the use of the schema collection S and relies only on the class descriptions D. It creates single-column pseudo-tables in Step 2 of Algorithm 1, without leveraging the multi-column structure that ZTab uses during fine-tuning.

Baselines and References: Zero-shot models, ArcheType$_{ZS}$ (Feuer et al., 2024), GPT$_{col\text{-}ZS}$, GPT$_{table\text{-}ZS}$ (Korini & Bizer, 2023), and Chorous (Kayali et al., 2024), serve as our *baselines*, as they share the same zero-shot constraint as ZTab. In addition, we include few-shot models (GPT$_{col\text{-}FS}$, GPT$_{table\text{-}FS}$ (Korini & Bizer, 2023)) and supervised models (Doduo (Suhara et al., 2022), ArcheType$_{FT}$ (Feuer et al., 2024)) as *references* to highlight the performance gap between ZTab and methods that rely on labeled training data. These reference models are not considered baselines, as they fall outside the zero-shot setting.

5. Results

Table 1 details the comparison on individual datasets. ZTab outperforms the best zero-shot model (GPT$_{table\text{-}ZS}$) by 10.4% and outperforms the best few-shot model (GPT$_{table\text{-}FS}$) by 7%. While supervised models slightly outperform ZTab due to their access to complete user-provided training data, ZTab operates without depending on any user-provided tabular training data same as zero-shot models.

Roughly speaking, the improvement of ZTab over zero-shot baselines increases as the number of classes in the dataset increases. This highlights a common difficulty of existing LLM based zero-shot models in dealing with datasets with a large number of classes where semantically similar classes can confuse the model. ZTab addresses this ambiguity by providing a few examples of each class via class descriptions, demonstrating its ability to resolve class overlap and improve accuracy.

Compared to the zero-shot baselines, ZTab pays in fine-tuning time: approximately 1 hour for T2D, and approximately 8, 12, and 18 hours for SOTAB$_{sch\text{-}s}$, SOTAB$_{dbp}$, and SOTAB$_{sch}$, respectively. This cost is justifiable by the large performance gain (at least 10.4%) over zero-shot baselines.

ZTab outperforms ZTab$_{w/o\ S}$ by leveraging both class descriptions and the table schema collection. This allows the model to utilize the global context of the table during prediction, leading to improved performance across all datasets.

6. Conclusion

We presented ZTab, a novel zero-shot framework for column type annotation designed to overcome the limitations of existing zero-shot models. ZTab has the performance close to fully supervised models but requires no labeled training data, similar to zero-shot learning. These performances are achieved by using two LLMs, one LLM is used for generating pseudo-fine-tuning tables from provided semantic types and table schemas, and one open-source LLM is fine-tuned using the pseudo-tables, similar to supervised learning. This fine-tuning allows the LLM to effectively learn the structure of tabular data and distinguish between similar classes.

References

Sotab dataset, 2023. URL https://webdatacommons.org/structureddata/sotab/v2/.

Bai, J., Bai, S., Chu, Y., Cui, Z., Dang, K., Deng, X., Fan, Y., Ge, W., Han, Y., Huang, F., et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Berkovitch, Y., Koutrika, G., Kraska, T., and Kim, H. J. Generating tables from the parametric knowledge of language models. *arXiv preprint arXiv:2406.10922*, 2024.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chen, J., Jiménez-Ruiz, E., Horrocks, I., and Sutton, C. Learning semantic annotations for tabular data. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pp. 2088–2094, 2019.

Deng, X., Sun, H., Lees, A., Wu, Y., and Yu, C. Turl: Table understanding through representation learning. *ACM SIGMOD Record*, 51(1):33–40, 2022.

Fernandez, R. C., Abedjan, Z., Koko, F., Yuan, G., Madden, S., and Stonebraker, M. Aurum: A data discovery system. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pp. 1001–1012. IEEE, 2018a.

Fernandez, R. C., Mansour, E., Qahtan, A. A., Elmagarmid, A., Ilyas, I., Madden, S., Ouzzani, M., Stonebraker, M., and Tang, N. Seeping semantics: Linking datasets using word embeddings for data discovery. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pp. 989–1000. IEEE, 2018b.

Feuer, B., Liu, Y., Hegde, C., and Freire, J. Archetype: A novel framework for open-source column type annotation using large language models. *Proceedings of the VLDB Endowment*, 17(9):2279 – 2292, 2024.

Hai, R., Koutras, C., Quix, C., and Jarke, M. Data lakes: A survey of functions and systems. *IEEE Transactions on Knowledge and Data Engineering*, 35(12):12571–12590, 2023.

Hoseinzade, E. and Wang, K. Graph neural network approach to semantic type detection in tables. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 121–133. Springer, 2024.

Hulsebos, M., Hu, K., Bakker, M., Zgraggen, E., Satyanarayan, A., Kraska, T., Demiralp, Ç., and Hidalgo, C. Sherlock: A deep learning approach to semantic data type detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1500–1508, 2019.

Kandel, S., Paepcke, A., Hellerstein, J., and Heer, J. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 3363–3372, 2011.

Kayali, M., Lykov, A., Fountalis, I., Vasiloglou, N., Olteanu, D., and Suciu, D. Chorus: Foundation models for unified data discovery and exploration. *Proceedings of the VLDB Endowment*, 17(8):2104–2114, 2024.

Korini, K. and Bizer, C. Column type annotation using chatgpt. *arXiv preprint arXiv:2306.00745*, 2023.

Korini, K. and Bizer, C. Column property annotation using large language models. *Crete, Greece*, 2024.

Li, P., He, Y., Yashar, D., Cui, W., Ge, S., Zhang, H., Rifinski Fainman, D., Zhang, D., and Chaudhuri, S. Table-gpt: Table fine-tuned gpt for diverse table tasks. *Proceedings of the ACM on Management of Data*, 2(3):1–28, 2024.

Limaye, G., Sarawagi, S., and Chakrabarti, S. Annotating and searching web tables using entities, types and relationships. *Proceedings of the VLDB Endowment*, 3(1-2): 1338–1347, 2010.

Rahm, E. and Bernstein, P. A. A survey of approaches to automatic schema matching. *the VLDB Journal*, 10(4): 334–350, 2001.

Suhara, Y., Li, J., Li, Y., Zhang, D., Demiralp, Ç., Chen, C., and Tan, W.-C. Annotating columns with pre-trained language models. In *Proceedings of the 2022 International Conference on Management of Data*, pp. 1493–1503, 2022.

Sui, Y., Zhou, M., Zhou, M., Han, S., and Zhang, D. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pp. 645–654, 2024.

Sun, Y., Xin, H., and Chen, L. Reca: Related tables enhanced column semantic type annotation framework. *Proceedings of the VLDB Endowment*, 16(6):1319–1331, 2023.

Zhang, D., Suhara, Y., Li, J., Hulsebos, M., Demiralp, Ç., and Tan, W.-C. Sato: Contextual semantic type detection in tables. *arXiv preprint arXiv:1911.06311*, 2019.

A. Analysis

In this section, we analyze the effect of different components of ZTab, namely, class description size e, schema sampling rating r, prompt design, base LLM M, Description LLM M_d, and row size k.

Class Description Size: Table 3 presents the performance of ZTab under varying class description sizes e, i.e., 500 (i.e., All), 50, 25, 12, and 6. The best performance is achieved with the full size, as more examples of classes lead to more diverse pseudo-training tables, which improves the model's ability to generalize. However, ZTab demonstrates robust performance even with as few as 6 examples per class, by leveraging the extensive knowledge encoded in LLM's pre-training on large textual corpora. For the best performance, we recommend the full class description size $e = 500$ for more example diversity.

	Class Description Size				
Dataset	All	50	25	12	6
$\text{SOTAB}_{\text{sch}}$	**78.0**	76.9	76.4	75.7	75.1
$\text{SOTAB}_{\text{sch-s}}$	**76.2**	74.2	74.3	73.5	72.8
$\text{SOTAB}_{\text{dbp}}$	**78.5**	76.4	76.2	75.3	75.1
T2D	**96.2**	95.9	95.4	95.6	95.2

Table 3. Micro-F1 score for ZTab with different class description sizes e.

Prompt Design: We explore alternative table presentations and prediction methods of the **PromptConstruction** function. The table presentation can be either **column-by-column** or **row-by-row** (see Fig 2), and the prediction method can be either predicting **all columns together** or predicting **one target column** at a time. Table 4 compares the performance of ZTab with these alternative prompt designs.

Table:

Column 1	Column 2	...	Column n
t_{11}	t_{21}	...	t_{n1}
t_{12}	t_{22}	...	t_{n2}
...			
t_{1k}	t_{2k}	...	t_{nk}

Figure 2. Row-by-row presentation of a table with n columns and k rows.

The best performance is observed with the column-by-column presentation and target column prediction. The column-by-column presentation allows ZTab to focus on the context of each column individually, which simplifies the learning and leads to more accurate results because the values within each column present examples of the same semantic type. In contrast, the row-by-row presentation introduces values of different semantic types on each row, which makes it harder for the row-based reading to capture the relationships between columns. When predicting all columns together, ZTab's performance tends to decrease, particularly when using smaller annotation LLMs like Mistral, because the model may generate an incorrect number of semantic types for a table (e.g., predicting four or six types for a table with five columns). Furthermore, even if ZTab detects the correct semantic types, it may not align them correctly with the corresponding columns.

Present	Predict	$\text{SOTAB}_{\text{sch}}$	$\text{SOTAB}_{\text{sch-s}}$	$\text{SOTAB}_{\text{dbp}}$	T2D
col-by-col	target	**78.0**	**76.2**	**78.5**	**96.2**
col-by-col	all	74.0	73.5	75.2	94.4
row-by-row	target	74.3	74.1	74.3	95.1
row-by-row	all	71.5	71.1	72.0	92.4

Table 4. Micro-F1 score for ZTab using different prompt designs.

Annotation Models M_a: Table 5 compares the performance of ZTab based on various open-source annotation LLMs M_a: Doduo (BERT, 110M parameters), Phi3-mini (3.8B parameters), LLaMA3 (8B parameters), Mistral (7B parameters), and Qwen (7B parameters). To have full control over the fine-tuning process, we only utilize open-source LLMs as the annotation models. In general, larger models achieve higher micro-F1 scores. In the absence of user-provided training data

and when relying on small examples of semantic types (i.e., class descriptions), a powerful annotation LLM helps capture general knowledge needed for accurate column type annotation in the zero-shot setting. Compared to the zero-shot baselines in Table 1, which use the closed-source GPT-3.5-Turbo-1106 model (whose parameter count is unpublished), the ZTabs based on the above open-source and probably smaller models (except for Doduo) are highly competitive due to the effective fine-tuning using the pseudo-data generated by our approach.

Dataset	**Qwen**	Mistral	LLama3	Phi3	Doduo
$\text{SOTAB}_{\text{sch}}$	**78.0**	75.3	73.1	70.0	51.5
$\text{SOTAB}_{\text{sch-s}}$	**76.2**	74.1	71.9	68.7	49.7
$\text{SOTAB}_{\text{dbp}}$	**78.5**	75.2	68.6	70.1	45.3
T2D	**96.2**	93.5	86.4	90.7	74.4

Table 5. Micro-F1 score for ZTab of different base models M.

Schema Sampling Ratio: Table 6 shows the number of schemas and unique schemas of different datasets. Table 7 shows how ZTab performs with different schema sampling ratios r. We consider the three SOTAB datasets that have a large table schema collection S.

The worst performance occurs at the small sampling ratio of 1% due to too few schemas used in each epoch. Increasing to 2.5% improves the results but a further increase provides little additional benefit. This is because many schemas in S are redundant (see Table 6), sampling all of them is unnecessary and the 2.5% sampling ratio captures enough variety of schemas over the specified number of epochs and provides wide and most likely all class coverage.

Dataset	**#Schemas**	**#Unique Schemas**
T2D	160	64
$\text{SOTAB}_{\text{sch}}$	44,769	4,189
$\text{SOTAB}_{\text{sch-s}}$	10,631	1,643
$\text{SOTAB}_{\text{dbp}}$	37,631	1,780

Table 6. Number of total and unique schemas in SOTAB datasets.

	Schema sampling ratio				
Dataset	10%	7.5%	5%	2.5%	1%
$\text{SOTAB}_{\text{sch}}$	77.5	77.3	76.9	78.0	74.6
$\text{SOTAB}_{\text{sch-s}}$	76.4	75.8	75.9	76.2	73.8
$\text{SOTAB}_{\text{dbp}}$	78.2	77.9	77.3	78.5	74.3

Table 7. Micro-F1 score for ZTab using different schema sampling ratios r.

Description Model M_d: In all the experiments, ZTab employs ChatGPT-3.5 as the description model M_d. Both open-source and closed-source LLMs can be used for M_d, however, stronger LLMs (e.g., GPT-3.5) are preferred to ensure high-quality and semantically rich class descriptions.

Row Size k: Until now, all experiments for ZTab are based on $k = 3$, i.e., all pseudo-tables have 3 rows. Larger k values add little benefit in performance but increases computational cost.