

LLM Knows, Speaks Not: Extracting Decoder-State Representations for Column Type Annotation

Lingxi Cui^{1,2*}, Jianfeng Zhang^{1,2*}, Huan Li^{1,2}, Ke Chen^{1,2}, Lidan Shou^{1,2✉}

¹The State Key Laboratory of Blockchain and Data Security, Zhejiang University

²Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security
{cuilingxi.cs, jianfeng.zhang, lihuan.cs, chenk, should}@zju.edu.cn

Abstract

Column Type Annotation (CTA) enriches tables with semantic column types that are often missing or incomplete in real-world data, aiding multiple downstream table tasks. Yet CTA remains challenging because sparse or noisy column values often require table-level context for disambiguation. Prior work either flattens tables into encoder inputs, losing structure and semantics, or follows a description-then-encoding pipeline that separates encoding and adds generation cost. We propose **D2R**, a decoder-based framework that obtains column representations directly from the hidden state that initiates column description generation. Specifically, D2R first encodes the table context once as a shared prefix KV cache, providing table-level context for all target columns. For each column, we append its sampled values and a description trigger token, then extract the trigger’s hidden state as the column embedding, preserving semantics without producing text. A decoder-attached classifier, trained via contrastive learning aligned with the CTA objective, predicts column types directly from this embedding. Across four CTA benchmarks, D2R achieves the best Micro-F1, outperforming TURL, a strong table PLM, by up to 12.57 points, while retaining efficient inference. The code is available at <https://github.com/IvesCheung/cta>.

1 Introduction

Tables serve as a ubiquitous medium for organizing and managing data across diverse domains, often in a structured and text-heavy form (Cui et al., 2026; Fan et al., 2023b). As the volume of tabular data grows, the lack of consistent and machine-readable column semantics, specifically column names and their semantic types, becomes a major bottleneck for automated data processing (Ding et al., 2025).

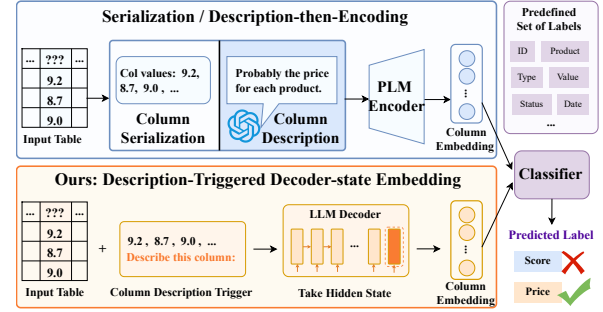


Figure 1: **Top:** prior CTA paradigms either serialize columns into an encoder or describe them via an LLM and then encode separately. **Bottom:** our approach D2R avoids the separate encoding step by extracting the representation directly from the decoder’s description-triggered hidden state.

This motivates *Column Type Annotation* (CTA): the task of extracting semantic type labels (e.g., price, rating, or address) for target columns from their cell values and surrounding table context, without assuming reliable metadata (Zhang et al., 2020; Suhara et al., 2022; Miao and Wang, 2023). These extracted column types provide structured semantic metadata for downstream tasks such as schema matching (Zhang et al., 2024a; Du et al., 2024), knowledge-base construction (Gottschalk and Demidova, 2022; Kruit et al., 2020), and table integration (Khawwaja et al., 2023).

CTA remains challenging because column values are often ambiguous without **table-level context** (see Figure 1 Top): values such as 9.2, 8.7, and 9.0 may indicate a rating, score, or price depending on neighboring columns. Existing methods mainly follow two paradigms. *Serialization-then-encoding* methods (Deng et al., 2020; Suhara et al., 2022; Fan et al., 2023b; Miao and Wang, 2023; Sun et al., 2023) flatten a table or column into text and feed it to a PLM encoder. However, flattening weakens table structure and produces incoherent text fragments, thus limiting the model’s ability to capture fine-grained **column semantics**. *Description-*

*Equal contribution. ✉Corresponding author.

then-encoding methods (Zhang et al., 2025; Balaka et al., 2025) instead ask an LLM to summarize the column and encode the generated summarization, yielding richer column semantics. However, they require **extra generation**, depend on description quality, and decouple the representation from direct CTA supervision.

This observation motivates the central question of this paper: *When an LLM is instructed to describe a table column, does its decoder state evolve into a qualified representation of that column?* If the answer is yes, then CTA could benefit from LLM-style semantic abstraction without incurring a separate description encoder.

To answer this question, we propose a decoder-based framework called D2R (**D**escription-triggered **D**ecoder-state **R**epresentation) that turns the state prepared for column description generation into the representation used for CTA. Specifically, D2R first feeds the table context into the decoder and stores a reusable prefix key-value (KV) cache. For each target column, a suffix containing sampled values and a description trigger¹ is processed against this cache, allowing the suffix query to attend to semantically relevant columns and resolve ambiguity through cross-column context. To retain the semantic benefits of a column description without actually generating and re-encoding text, D2R extracts the hidden state at the description trigger’s last token from an intermediate decoder layer as the column embedding, preserving the decoder’s continuous semantic state before it is compressed into discrete text. Finally, this embedding is passed to a multi-layer classification head attached to the decoder and optimized with supervised contrastive learning, directly aligning the decoder-derived representation with the CTA objective.

Contributions. We make the following contributions: **(1) Description-triggered representation.** We introduce a new CTA framework that uses the description trigger token as a semantic anchor to distill a column representation directly from the decoder’s hidden state, unifying description-level semantics without separate generation and re-encoding. **(2) Table context via cached attention.** A shared table-prefix KV cache and target-column suffix let each column attend to table-level and cross-column evidence. **(3) Decoder-attached classification head.** We attach a multi-layer clas-

sifier directly to the decoder state and train it with supervised contrastive learning, directly aligning the extracted representation with the CTA objective. **(4) Empirical validation.** Across four CTA benchmarks, D2R delivers consistent improvements over strong baselines, outperforming TURL by up to 12.57 points, while maintaining efficient inference.

2 Preliminaries

2.1 Problem Definition

Column Type Annotation (CTA) aims to infer the semantic type (e.g., price, rating) of a target column from a predefined set of labels, which is fundamental for understanding and integrating tabular data. We consider CTA scenarios without metadata (e.g., column names), where the prediction must rely only on cell values and table context. This setting is more general and challenging. Column names are often missing, ambiguous, or mismatched in practice (Fan et al., 2023a), and prior works adopt the same configuration (Ding et al., 2025).

Definition 1 (Column Type Annotation (CTA))

Let a table T consist of n columns $\{c_i\}_{i=1}^n$. Given a label space $\mathcal{Y}$ and a target column c_i , the goal is to learn a mapping

$$f(T, c_i) \rightarrow y_i, \quad y_i \in \mathcal{Y}, \quad (1)$$

that predicts the semantic type of c_i based on its own values and the table context.

2.2 Related Work

Column Type Annotation. CTA has progressed from knowledge-base matching (Venetis et al., 2011; Limaye et al., 2010) to neural models that exploit neighboring-column context (Zhang et al., 2020). Most modern CTA systems follow a serialization-then-encoding pipeline: they serialize a target column or an entire table into text and feed it to a pretrained encoder, often adapted with supervised fine-tuning (TURL (Deng et al., 2020)), contrastive pretraining (Starmie (Fan et al., 2023b) and Watchdog (Miao and Wang, 2023)), or stronger context retrieval (REVEAL (Ding et al., 2025)). LLMs have also been used as direct predictors (Korini and Bizer, 2023; Feuer et al., 2024), but a more relevant LLM-assisted paradigm first generates a natural language description of the target column and then encodes it with a PLM-based encoder (Ding et al., 2025; Balaka et al., 2025), injecting richer column semantics into the representation stage. D2R is closest to this latter line, but

¹The description trigger is an instruction that prompts the decoder to generate column descriptions (see Appendix A.4).

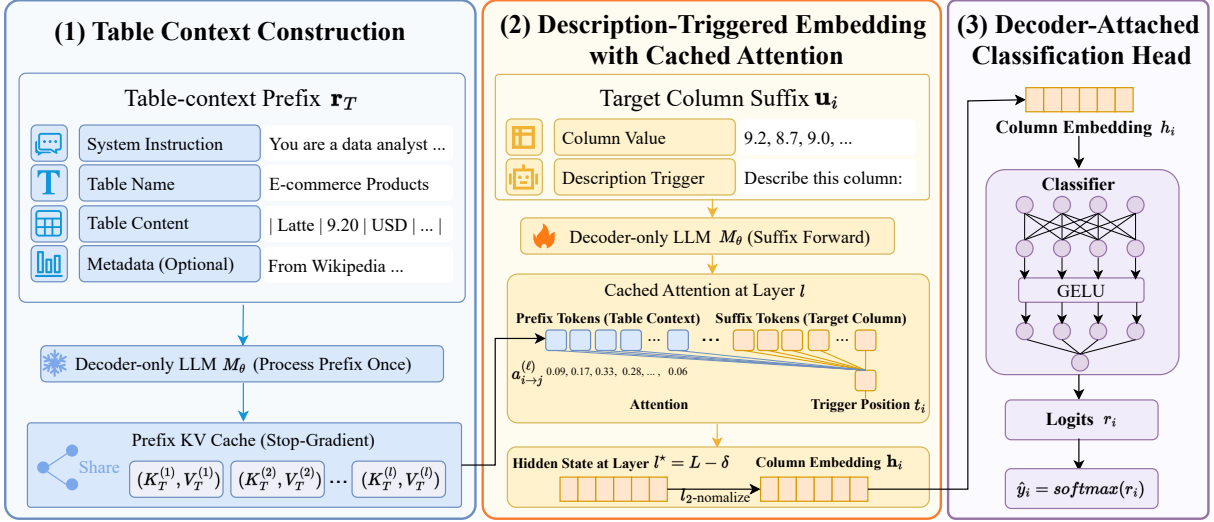


Figure 2: Overview of D2R. The table context is cached once as shared prefix context; each column suffix reuses this cache, and the description-triggered hidden state serves as the column representation for classification.

uses the decoder state that initiates the description instead of re-encoding generated text.

Designated-Token Representations. Probing studies (Voita et al., 2019; Skea et al., 2025) show that intermediate transformer layers often provide stronger semantic representations than the final layer, which is more directly optimized for next-token prediction. This motivates using selected token positions as task-specific signals. PET (Schick and Schütze, 2021) uses textual triggers to turn the language-model output at a designated prediction position into label scores through verbalizers. Causal2Vec (Lin et al., 2026) appends a contextual token to the decoder and pools the last-position hidden state as a continuous representation. D2R differs by placing the special token at the point where the decoder begins column-description generation and supervising this state directly for CTA.

3 Methodology

3.1 Overview

Representation for CTA. Column type annotation (CTA) is typically cast as a classification problem over column embeddings c_i :

$$\hat{y}_i = \text{CLS}(\mathbf{h}_i), \quad \mathbf{h}_i = \text{GetEmbedding}(T, c_i).$$

Existing pipelines instantiate GetEmbedding in two dominant ways: serialization-then-encoding or description-then-encoding. Once the column embedding $\mathbf{h}_i$ is obtained, a classification head (e.g., MLP) is applied to predict the column type.

Decoder State as Representation. D2R instead obtains $\mathbf{h}_i$ directly from the decoder’s internal state at the moment it would begin generating a natural language description². This avoids key limitations of prior work: direct serialization ignores table structure and lacks semantics, while generating a description separately adds cost and discards the decoder’s rich continuous state. By capturing the representation at the boundary of description generation, the decoder has already parsed the table context, resolved ambiguities, and prepared a semantically rich, highly discriminative, and differentiable column embedding.

Framework. As illustrated in Figure 2, D2R proceeds in three stages:

(1) *Table Context Construction* (Section 3.2): The table context is fed once to the decoder-only language model, and the resulting KV cache is stored and detached from gradients.

(2) *Description-Triggered Embedding with Cached Attention* (Sections 3.3 and 3.4): Each target-column suffix contains sampled values and a description trigger. The hidden state at the trigger position is extracted and normalized as the column embedding $\mathbf{h}_i$. During the suffix forward pass, standard cached self-attention lets the target column retrieve evidence from the table prefix.

(3) *Decoder-attached Classification Head* (Section 3.5): The embedding is passed to a lightweight MLP classification head attached to the decoder,

²This design is later validated in Section 4.3, where we compare it against embeddings produced by a same-sized encoder model under the same classifier.

and the whole framework is optimized with a supervised contrastive objective.

3.2 Table Context Construction

A column’s type is often under-determined by its values alone. For instance, the same list of numbers could represent prices, populations, or scores depending on the surrounding columns. Humans resolve this ambiguity by scanning the table for cues such as units, currencies, and neighboring attributes. D2R gives the decoder an analogous ability without repeatedly encoding the whole table: we construct a shared table-context prefix and cache its key-value states once.

The table-context prefix $\mathbf{r}_T$ concatenates the system instruction, table name, partial table content, and optional metadata, with missing components set to empty strings. Subsequently, given the decoder-only language model $\mathcal{M}_\theta$ with L layers, the prefix $\mathbf{r}_T$ is processed only once to obtain the table-context KV cache:

$$\mathcal{C}_T = \left\{ \left(\mathbf{K}_T^{(\ell)}, \mathbf{V}_T^{(\ell)} \right) \right\}_{\ell=1}^L = \text{KVCache}_{\mathcal{M}_\theta}(\mathbf{r}_T). \quad (2)$$

During training, the prefix cache is detached $\bar{\mathcal{C}}_T = \text{sg}(\mathcal{C}_T)$, where $\text{sg}(\cdot)$ denotes stop-gradient, so that gradients are concentrated on the shorter per-column suffix pass and the task heads. At inference time, the same cache can be reused for all target columns in the table.

3.3 Description-Triggered Embedding

To obtain a single vector that preserves the decoder’s internal description-level semantics for a column, this section constructs the target-column suffix and extracts the hidden state at the description trigger token as the column embedding.

For target column c_i , we construct the column suffix as:

$$\mathbf{u}_i = \rho(i) \oplus \eta(c_i) \oplus \gamma, \quad (3)$$

where $\rho(i)$ is a column identifier, $\eta(\cdot)$ serializes the sampled non-empty values of the column (sampling at most a preset number while preserving row order), γ is a fixed description trigger (e.g., provide the semantic type and meaning of this column), and “ $\oplus$ ” denotes string concatenation. The trigger γ cues the decoder to verbalize the column’s semantics, but we do not let the decoder complete the generation. Instead, we extract the hidden state at the last trigger token $t_i = |\mathbf{u}_i|$, i.e.,

the position immediately before the decoder would produce its first output token.

With the shared cache from Section 3.2, the suffix forward pass yields hidden states:

$$\left\{ \mathbf{x}_{i,t_i}^{(\ell)} \right\}_{\ell=1}^L = \mathcal{M}_\theta(\mathbf{u}_i; \bar{\mathcal{C}}_T), \quad (4)$$

where $\mathbf{x}_{i,t_i}^{(\ell)}$ is the hidden state of the last trigger token t_i at layer ℓ . We extract the hidden state at an intermediate layer $\ell^* = L - \delta$ and ℓ_2 -normalize it as the column embedding:

$$\mathbf{h}_i = \frac{\mathbf{x}_{i,t_i}^{(\ell^*)}}{\max(\|\mathbf{x}_{i,t_i}^{(\ell^*)}\|_2, \varepsilon)}, \quad \ell^* < L. \quad (5)$$

We call $\mathbf{h}_i$ the *description-triggered embedding* because it is captured at the exact moment the decoder is poised to generate a description. The choice of ℓ^* is motivated by how decoder layers divide labor. Lower layers capture surface patterns, the final layer is tuned toward next-token prediction, and intermediate layers tend to carry abstract semantic content before it is projected back into vocabulary space (Voita et al., 2019; Schick and Schütze, 2021; Skean et al., 2025; Lin et al., 2026). By reading the state at $L - \delta$ rather than L , we extract the representation after the column’s meaning has been consolidated but before it is mapped to a specific output distribution. The ablation in Section 4.3 confirms this empirically.

3.4 Cached Attention for Table Context

To provide table context and resolve ambiguity for the column representation, the suffix forward pass in Section 3.3 reuses the table context from Section 3.2, as shown in Figure 2 (2).

At each decoder layer, each suffix token’s query attends to the cached table-context prefix as well as to the current suffix tokens. Let $\mathbf{o}_{i,t}^{(\ell)}$ denote the attention output for the t -th token of the suffix at layer ℓ . Define $\tilde{\mathbf{K}}_{i,t}^{(\ell)} = [\bar{\mathbf{K}}_T^{(\ell)}; \mathbf{K}_{i,\leq t}^{(\ell)}]$ and $\tilde{\mathbf{V}}_{i,t}^{(\ell)} = [\bar{\mathbf{V}}_T^{(\ell)}; \mathbf{V}_{i,\leq t}^{(\ell)}]$. Then,

$$\mathbf{o}_{i,t}^{(\ell)} = \text{softmax} \left(\frac{\mathbf{q}_{i,t}^{(\ell)} \tilde{\mathbf{K}}_{i,t}^{(\ell)\top}}{\sqrt{d_k}} \right) \tilde{\mathbf{V}}_{i,t}^{(\ell)}. \quad (6)$$

Here, $\mathbf{q}_{i,t}^{(\ell)}$ is the query vector produced for the t -th token in suffix $\mathbf{u}_i$, while $\mathbf{K}_{i,\leq t}^{(\ell)}$ and $\mathbf{V}_{i,\leq t}^{(\ell)}$ are the keys and values of suffix tokens up to position t . The terms $\bar{\mathbf{K}}_T^{(\ell)}$ and $\bar{\mathbf{V}}_T^{(\ell)}$ are the detached prefix keys and values from $\bar{\mathcal{C}}_T$. The cached prefix

keys and values contain table context such as table name and other columns, so the suffix query can retrieve cross-column evidence that disambiguates the target column’s values.

This column relation signal can be written explicitly from the attention distribution. Let $\mathcal{P}_j$ be the prefix-token positions corresponding to source column c_j in the serialized table. At the description trigger position t_i , the attention mass from target column i to source column j is:

$$a_{i \rightarrow j}^{(\ell)} = \sum_{p \in \mathcal{P}_j} \alpha_{i, t_i, p}^{(\ell)}, \quad (7)$$

where $\alpha_{i, t_i, p}^{(\ell)}$ is the normalized attention weight assigned to prefix token p . Although no explicit relation supervision is imposed, this quantity measures how much evidence the cached attention routes from each source column to the target column. No auxiliary relation-classification loss or graph construction is needed, as the model learns which source columns are informative for each target purely from the CTA supervision signal. Our experiments (Section 4.4) further analyze whether the target column’s attention focuses on semantically related columns.

3.5 Decoder-Attached Classification Head

After obtaining $\mathbf{h}_i$, we attach the classifier directly to the decoder representation. Attaching the head at the description trigger position, rather than at the output of a separate encoder, means the classifier receives a representation that has already been conditioned on the full table context through prefix-suffix attention. An R -layer MLP maps the generation-time embedding to type logits, where $q = 1, \dots, R$:

$$\begin{aligned} \mathbf{r}_i^{(0)} &= \mathbf{h}_i, \\ \mathbf{r}_i^{(q)} &= \text{Dropout} \left(\sigma(\mathbf{W}^{(q)} \mathbf{r}_i^{(q-1)} + \mathbf{b}^{(q)}) \right), \\ \hat{\mathbf{y}}_i &= \text{softmax}(\mathbf{W}_{\text{out}} \mathbf{r}_i^{(R)} + \mathbf{b}_{\text{out}}). \end{aligned} \quad (8)$$

For a mini-batch $\mathcal{B}$, the classification loss is the standard cross-entropy:

$$\mathcal{L}_{\text{cls}} = -\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \log \hat{\mathbf{y}}_i[y_i]. \quad (9)$$

To further sharpen the embedding space and align the extracted representation with the downstream CTA objective, we add a supervised contrastive regularizer (Khosla et al., 2020). A two-layer projection head maps $\mathbf{h}_i$ to a normalized embedding $\mathbf{z}_i$ on the unit sphere. For anchor i , let

$P(i) = \{p \in \mathcal{B} \setminus \{i\} \mid y_p = y_i\}$ denote its positive set and let $\mathcal{I} = \{i \in \mathcal{B} \mid |P(i)| > 0\}$. The contrastive loss is:

$$\begin{aligned} \mathcal{L}_{\text{con}} = & -\frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \frac{1}{|P(i)|} \sum_{p \in P(i)} \left(\log \exp(\mathbf{z}_i^\top \mathbf{z}_p / \tau) \right. \\ & \left. - \log \sum_{a \in \mathcal{B} \setminus \{i\}} \exp(\mathbf{z}_i^\top \mathbf{z}_a / \tau) \right). \end{aligned} \quad (10)$$

The final loss is $\mathcal{L} = \mathcal{L}_{\text{cls}} + \lambda \mathcal{L}_{\text{con}}$.

4 Experiments

We conduct a thorough experimental evaluation to empirically address these questions: **Q1 (Section 4.2)**: Does D2R improve CTA accuracy on standard benchmarks? **Q2 (Section 4.3)**: Is the proposed description-triggered embedding better than embeddings from separate encoders, and how does it vary across hidden state extraction strategies and decoder backbones? **Q3 (Section 4.4)**: Does cached attention over the table-context prefix route useful column context, and how much prefix context is needed? **Q4 (Section 4.5)**: How much does the decoder-attached MLP classification head and contrastive learning contribute? **Q5 (Section 4.6)**: Is directly extracting column embeddings from the decoder state more efficient? **Q6 (Section 4.7)**: Is D2R robust under wide and noisy table schemas?

4.1 Experimental Setup

Datasets. We evaluate on four CTA benchmarks: GitTablesDB and GitTablesSC (Hulsebos et al., 2023), SOTAB-CTA (Korini et al., 2022), and WikiTables-CTA (Deng et al., 2020). Dataset details and statistics are provided in Appendix A.6.

Baselines. We compare against three groups of methods. 1) Serialization-then-encoding methods include TURL (Deng et al., 2020), Sato (Zhang et al., 2020), Doduo (Suhara et al., 2022), Starmie (Fan et al., 2023b), RECA (Sun et al., 2023), Watchog (Miao and Wang, 2023), and REVEAL (Ding et al., 2025). 2) Description-then-encoding methods include an AutoDDG-style baseline (Zhang et al., 2025), which follows AutoDDG’s description generation strategy, and Qwen-DescEmbed, a Qwen-based variant that uses Qwen2.5-72B-Instruct (Yang et al., 2024) to generate descriptions and Qwen3-Embedding (Yang et al., 2025) to encode them. 3) We further compare with methods that

Method	GitTablesDB		GitTablesSC		SOTAB-CTA		WikiTables-CTA	
	Micro	Macro	Micro	Macro	Micro	Macro	Micro	Macro
TURL	48.20 $\pm$ 0.98	19.56 $\pm$ 2.45	58.15 $\pm$ 0.98	26.26 $\pm$ 4.70	80.13 $\pm$ 0.25	77.34 $\pm$ 0.98	90.53 $\pm$ 0.07	66.40 $\pm$ 0.23
Sato	46.05 $\pm$ 0.87	23.50 $\pm$ 1.01	55.83 $\pm$ 0.34	30.07 $\pm$ 0.94	71.74 $\pm$ 0.25	70.01 $\pm$ 0.27	61.56 $\pm$ 0.84	32.05 $\pm$ 0.72
Doduo	44.77 $\pm$ 4.36	21.63 $\pm$ 3.26	52.36 $\pm$ 2.94	25.09 $\pm$ 5.01	81.32 $\pm$ 0.00	78.51 $\pm$ 0.00	92.19 $\pm$ 1.01	74.10 $\pm$ 0.86
Starmie	54.43 $\pm$ 0.93	30.71 $\pm$ 3.58	64.87 $\pm$ 2.47	37.04 $\pm$ 4.33	87.95 $\pm$ 0.27	86.84 $\pm$ 0.08	92.91 $\pm$ 0.08	76.45 $\pm$ 0.44
RECA	53.83 $\pm$ 1.67	25.70 $\pm$ 5.49	64.52 $\pm$ 1.69	35.10 $\pm$ 3.81	68.52 $\pm$ 0.35	68.21 $\pm$ 0.71	91.27 $\pm$ 0.03	73.36 $\pm$ 0.82
Watchog	53.96 $\pm$ 0.78	28.89 $\pm$ 3.34	65.24 $\pm$ 1.90	36.06 $\pm$ 3.39	86.23 $\pm$ 0.22	84.19 $\pm$ 0.06	92.25 $\pm$ 0.06	73.77 $\pm$ 0.35
REVEAL	59.79 $\pm$ 0.98	36.40 $\pm$ 3.17	<u>69.26 $\pm$ 0.93</u>	41.98 $\pm$ 1.96	88.35 $\pm$ 0.00	<u>87.60 $\pm$ 0.00</u>	<u>93.14 $\pm$ 0.51</u>	77.81 $\pm$ 0.43
AutoDDG-style	55.26 $\pm$ 1.68	32.82 $\pm$ 2.17	64.75 $\pm$ 1.72	43.12 $\pm$ 1.79	82.95 $\pm$ 0.31	79.53 $\pm$ 0.92	/	/
Qwen-DescEmbed	<u>59.93 $\pm$ 1.75</u>	<u>36.42 $\pm$ 2.15</u>	68.29 $\pm$ 1.41	42.01 $\pm$ 1.58	<u>88.40 $\pm$ 0.28</u>	84.60 $\pm$ 1.30	/	/
Qwen-Plus (5-shot)	57.48 $\pm$ 1.20	33.66 $\pm$ 1.57	63.21 $\pm$ 1.24	<u>44.81 $\pm$ 2.60</u>	52.61 $\pm$ 0.00	48.82 $\pm$ 0.00	32.21 $\pm$ 0.00	13.01 $\pm$ 0.00
TableLlama	11.03 $\pm$ 0.74	9.70 $\pm$ 1.53	12.85 $\pm$ 0.91	<u>11.71 $\pm$ 1.62</u>	23.35 $\pm$ 0.00	20.59 $\pm$ 0.00	91.85 $\pm$ 0.00	76.04 $\pm$ 0.00
D2R	60.77 $\pm$ 2.10	36.76 $\pm$ 2.98	69.56 $\pm$ 1.00	45.28 $\pm$ 3.52	88.79 $\pm$ 0.10	87.67 $\pm$ 0.28	93.46 $\pm$ 0.67	<u>77.59 $\pm$ 0.51</u>

Table 1: Main CTA results on four benchmarks. Scores are Micro-F1 and Macro-F1 (% , mean $\pm$ std); bold and underline indicate the best and second-best results, respectively.

directly prompt LLMs to predict column types, including Qwen-Plus in the 5-shot setting (Yang et al., 2024), as well as TableLlama (Zhang et al., 2024b).

Metrics. Following prior CTA work, we report Micro-F1 and Macro-F1. Micro-F1 reflects aggregate prediction accuracy, while Macro-F1 is more sensitive to rare semantic types. All tables report the mean over three random seeds under the same preprocessing and label splits.

Implementation Details. We implement D2R with Qwen3-0.6B and LoRA. The table-context prefix is encoded once into a reusable KV cache; all column suffixes are then processed together in a batched forward pass reusing this cache. The column embedding is obtained by averaging the last-token hidden states of layers $L - 4$ and $L - 8$, followed by L2 normalization. Contrastive weight $\lambda = 0.2$, temperature $\tau = 0.07$. See Appendix A.5 for full details (hardware, hyperparameters, etc.).

4.2 Q1: Main Results

Table 1 summarizes the main CTA comparison. **Overall strong performance.** D2R achieves the best Micro-F1 on all four benchmarks, improving over the strongest reported baseline by 0.84, 0.30, 0.39, and 0.32 points. Its Macro-F1 is best on GitTablesDB/SC and SOTAB-CTA, and second-best on WikiTables-CTA, showing that the description-triggered hidden state is a discriminative column representation. The strongest **serialization-then-encoding** baselines, Starmie and REVEAL, are precisely those that exploit contextual evidence. REVEAL further uses retrieval to select compact and informative column contexts. D2R instead lets the target suffix access table evidence through cached self-attention, while the description trigger encourages the decoder to organize column-

level semantics. This advantage is especially visible on the wide GitTablesDB/SC benchmarks, where limited encoder context makes it difficult to represent all relevant columns. **Description-then-encoding** baselines remain competitive, with Qwen-DescEmbed reaching 59.93/36.42 on GitTablesDB and 88.40/84.60 on SOTAB-CTA, confirming that LLM descriptions are semantically useful. However, this paradigm is not scalable: generating and re-encoding descriptions adds cost and is not reported on the 406K-table WikiTables-CTA. In contrast, D2R retains the semantic benefit without materializing text. **Direct LLM prompting is unstable.** Qwen-Plus drops sharply on SOTAB-CTA/WikiTables-CTA, and TableLlama is strong on WikiTables-CTA but fails on the other three datasets. This shows that simply asking a decoder to output column labels is unreliable for CTA tasks. Instead, our results validate a different use of decoder models: extracting the hidden state of a task-specific trigger token as the representation.

4.3 Q2: Description-Triggered Embedding

We analyze the proposed description-triggered embedding from multiple aspects, including the embedding source, decoder-state extraction strategy, the role of the description trigger, backbone sensitivity, and embedding-space quality.

Embedding Method	GitTablesDB		SOTAB-CTA	
	Micro	Macro	Micro	Macro
BERT-Embedding	50.06	20.98	81.81	74.39
Qwen-Embedding	45.40	19.23	79.46	73.97
Desc+Qwen-Embedding	59.93	36.42	88.40	84.60
D2R	60.77	36.76	88.79	87.67

Table 2: Comparison of embedding sources. All methods use the same MLP classifier and evaluation split. Scores are Micro-F1 and Macro-F1.

We compare different embedding sources (Table 2). Direct serialization-then-encoding embeddings are weaker: Qwen-Embedding (Qwen3-0.6B-embedding) achieves only 45.40/19.23 on GitTablesDB and 79.46/73.97 on SOTAB-CTA. BERT-Embedding (TaBERT (Yin et al., 2020)) performs slightly better due to table pretraining. With description-then-encoding (Desc+Qwen-Embedding), the scores improve to 59.93/36.42 on GitTablesDB and 88.40/84.60 on SOTAB-CTA, confirming that LLM-generated descriptions provide useful semantics. D2R achieves the best results on GitTablesDB (60.77/36.76) and on SOTAB-CTA (88.79/87.67), showing that the decoder’s truncated embedding captures description-level semantics comparable to description-then-encoding.

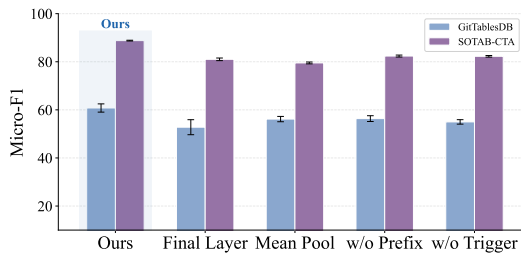


Figure 3: Ablation of decoder embedding extraction. Final layer uses the last decoder layer, mean pool averages all suffix-token states, w/o prefix removes cached table context, and w/o trigger removes the description trigger token from the suffix. Scores are Micro-F1.

Our final model averages the $L - 4$ and $L - 8$ hidden states of the last trigger token as the column embedding, always with the table-context prefix. Figure 3 ablates this strategy³. Intermediate trigger states outperform the final decoder layer, consistent with the final layer of a decoder being more tied to next-token prediction. We average the two layers because they behave similarly and give a more stable representation (see Appendix A.1). Compared to mean-pooling all suffix tokens, taking the last trigger token is more suitable for a causal decoder since the last token has attended to all preceding suffix tokens while earlier tokens lack full context. This differs from encoder-style pooling where bidirectional context makes averaging natural. Removing the table-context prefix weakens the embedding, confirming that cached table context provides useful evidence beyond the column itself. Removing the description trigger from the suffix also hurts accuracy, indicating that the trigger cues

³These variants are not expected to collapse, as the LoRA adapters and classifier are still trained with CTA supervision.

the decoder to capture holistic column semantics.

Variant	GitTablesDB		SOTAB-CTA	
	Micro	Macro	Micro	Macro
D2R (w/ trigger + w/o term)	60.77	36.76	88.79	87.67
w/ trigger + w/ term	57.33	32.14	86.16	84.54
w/o trigger + w/ term	56.42	29.71	84.78	79.92
w/ trigger + generated EOS	55.28	30.03	85.06	80.04

Table 3: Effect of descriptive semantics and the terminal readout token, under the same extraction layers and classifier. Scores are Micro-F1 and Macro-F1.

By default, D2R reads the intermediate-layer hidden state at the last token of the description trigger instruction without appending a dedicated terminal token. To disentangle descriptive semantics from an explicit terminal readout position, Table 3 compares four variants, where all terminal-token variants use the decoder’s own EOS token. D2R (w/ trigger + w/o term) reads the trigger’s last token. w/ trigger + w/ term appends EOS after the same trigger and reads the EOS state. w/o trigger + w/ term removes the trigger and appends EOS after the target-column values. w/ trigger + generated EOS generates the full column description and reads the final EOS state. With the same EOS readout, w/ trigger + w/ term outperforms w/o trigger + w/ term by 0.91/2.43 Micro/Macro-F1 on GitTablesDB and 1.38/4.62 on SOTAB-CTA, showing the benefit of descriptive semantics. Moreover, D2R without an added terminal token remains the best variant, showing that reading the last token of the description trigger is already sufficient. The generated-EOS variant performs worse, suggesting that the terminal state after full generation is oriented toward next-token prediction and thus carrying less column semantics than the intermediate state. Thus, reading the intermediate trigger state is both cheaper and more effective.

Backbone	GitTablesDB		SOTAB-CTA	
	Micro	Macro	Micro	Macro
Qwen3-0.6B	60.77	36.76	88.79	87.67
Qwen3-4B	61.02	38.51	88.77	87.08
Qwen3-8B	62.30	38.92	88.75	86.77
LLaMA3-3B	62.61	38.64	89.03	87.15

Table 4: Backbone sensitivity of D2R. Prompt format, extraction rule, and classifier are fixed.

Table 4 tests backbone sensitivity. On GitTablesDB, larger models improve consistently (Micro-F1: 60.77→61.02→62.30). On SOTAB-CTA, results stay close, likely because the dataset has small margins, performance is near saturation,

and larger backbones use smaller LoRA ranks to balance trainable parameters. LLaMA3-3B also performs strongly, confirming the framework generalizes across decoder families.

Finally, Figure 4 visualizes the embedding space before and after fine-tuning. Before fine-tuning, different labels are heavily mixed. After fine-tuning, same-label points form more compact and separated clusters (e.g., *id*, *type*, *status*, *state* move into clearer regions), and the overall geometry becomes more aligned with ground-truth labels. This confirms that the decoder’s hidden state, under supervised adaptation, becomes a highly discriminative representation for CTA.

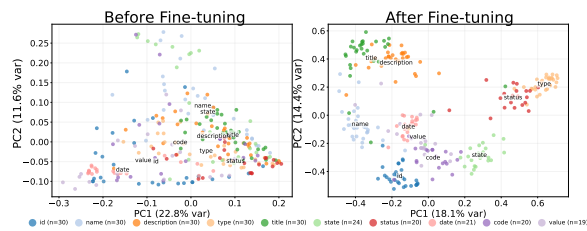


Figure 4: PCA visualization of column embeddings before and after fine-tuning, colored by ground-truth labels. The plot shows the top-10 labels in the GitTablesDB test split with at most 30 columns per label.

4.4 Q3: Cached Attention for Table Context

We examine cached attention via attention visualization and the effect of prefix width.

Figure 5 visualizes the attention mass defined in Eq. 7, the attention weight from the target column to each other column in the cached prefix. For the target column *price*, the highest attention falls on semantically correlated columns such as *currency* and *discount*, while unrelated columns receive little attention. For *product*, the model attends strongly to *category*, demonstrating that the prefix attention retrieves evidence from across the table, not merely from immediate neighbors. This confirms that cached attention routes useful column context into the representation of the target column.

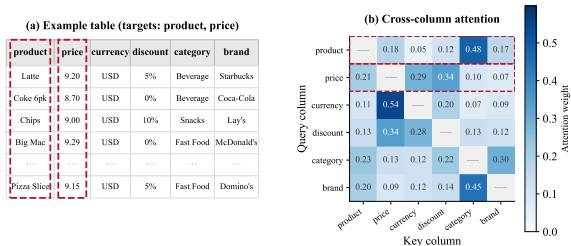


Figure 5: Cross-column attention heatmap.

We vary the number of columns in the cached prefix to test how much context is beneficial. Figure 6 shows an arch-shaped trend: too few columns provide insufficient contextual evidence, while too many columns make the prefix long and noisy, diluting the target-column suffix under finite attention capacity. The optimal prefix width differs across datasets, reflecting their schema width and column co-occurrence patterns. This suggests adaptively selecting informative prefix columns (e.g., via attention scores) could further improve efficiency and accuracy, which we leave for future work.

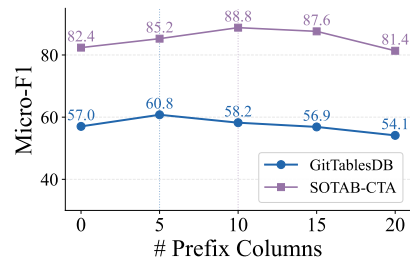


Figure 6: Performance with different numbers of columns in the cached prefix. Scores are Micro-F1.

4.5 Q4: Classification Head

Table 5 ablates the classifier head. Plain heads (1-layer linear or 2-layer MLP) perform poorly, and adding layer normalization (LN) improves results, showing that the raw decoder state is not directly separable by an unnormalized head. Incorporating CL yields further gains of 4.01/3.80 and 4.67/5.39 respectively, demonstrating that contrastive learning sharpens the embedding space. A residual MLP gives comparable scores, showing a lightweight, well-regularized head suffices.

Head variant	GitTablesDB		SOTAB-CTA	
	Micro	Macro	Micro	Macro
1-layer linear head	17.02	0.42	2.35	0.34
2-layer MLP	23.10	0.38	1.76	0.07
LN+2-layer MLP w/o CL	56.76	32.96	84.12	82.28
LN+2-layer MLP (D2R)	60.77	36.76	88.79	87.67
2-layer residual MLP	60.84	36.86	88.01	87.99

Table 5: Classification-head ablation. All variants attach the classifier directly to the same decoder embedding and differ only in head structure or auxiliary objective.

4.6 Q5: Efficiency Analysis

Figure 7 compares latency and accuracy on GitTablesDB. Serialization-then-encoding methods (e.g., Qwen-Embedding) are fast but low in accuracy, as they fail to capture column seman-

tics. Description-then-encoding (e.g., Desc+Qwen-Embedding) gains accuracy by first generating a natural language summary of the column values, but at the cost of high latency (521.04 ms/table) due to an extra LLM generation and a separate encoder pass. Our D2R (Qwen3-0.6B backbone) extracts the description-triggered hidden state during decoder forward, avoiding full text generation and re-encoding, thus preserving semantics while maintaining efficiency (14 ms/table). Moreover, D2R is even slightly faster than a model of the same size (Qwen-Embedding), since we stop at the hidden state instead of completing the generation.

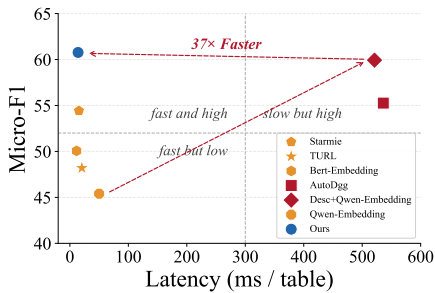


Figure 7: Latency-accuracy comparison on GitTablesDB across three paradigms: serialization-then-encoding (orange), description-then-encoding (red) and ours (blue). Scores are Micro-F1.

Method	Throughput (cols/s)	Avg. generated tokens	Avg. KV cache (GB/col)	Peak GPU memory (GB)
Qwen-DescEmbed	4.78	32.0	0.065	3.11
D2R	16.14	–	0.077	3.12

Table 6: Efficiency of D2R versus Qwen-DescEmbed on GitTablesDB. For a scale-matched comparison, Qwen-DescEmbed uses Qwen3-0.6B for generation and Qwen3-Embedding-0.6B for encoding, rather than the Qwen2.5-72B-Instruct generator used in the main experiments and Figure 7.

To further quantify the computational cost, we report throughput, the average KV cache per column, and peak GPU memory (Table 6). D2R achieves $3.4\times$ higher column throughput than Qwen-DescEmbed by directly using hidden states from a single forward pass instead of generating and re-encoding textual descriptions. On average, Qwen-DescEmbed generates 32 tokens per column before encoding. The per-column KV cache of D2R is only about 0.012 GB larger than that of Qwen-DescEmbed, since generating a description also requires the surrounding table context as input. This shows that reusing the prefix KV cache across columns does not introduce a noticeable memory

overhead. As a result, the peak GPU memory consumption of the two methods is almost identical.

4.7 Q6: Robustness

For robustness under wide and noisy schemas, we conduct a controlled study on GitTablesDB and SOTAB-CTA (Table 7). To introduce schema noise, we randomly drop 10–30% of the surrounding columns and inject 1–2 empty values into the 5 sampled cell values of each target column. For wide schemas, we select the top 10% of tables ranked by the number of columns on each benchmark.

Setting	GitTablesDB			SOTAB-CTA		
	Avg.Cols	Micro	Macro	Avg.Cols	Micro	Macro
Original	12.10	60.77	36.76	8.10	88.79	87.67
Schema noise	9.84	58.87	35.23	6.33	87.17	85.90
Wide tables	39.26	57.05	33.58	18.11	82.23	79.00

Table 7: Robustness of D2R under wide and noisy schemas, with the original test set as a reference.

Across both settings, D2R remains reliable relative to its performance on the original test set. We attribute this stability to the semantic understanding of the LLM, which gives D2R some resistance to noise. For example, even when the currency column next to a price column is dropped, the decoder can still infer from a neighboring discount column that the target column is a price. This suggests that D2R does not rely on any single surrounding column and remains reliable when the schema is partially corrupted. The drop is slightly larger under wide tables, which we attribute to our use of a fixed number of surrounding columns (5 by default) when sampling the table context, a window that covers only a small fraction of the columns on very wide tables (up to 39 columns on average).

5 Conclusion

We introduce a new paradigm for column type annotation that directly extracts the decoder’s hidden state at the description trigger token as the column representation. This simple yet powerful design effectively exploits a decoder language model’s innate ability to understand column semantics and integrate table context. Extensive experiments across four benchmarks demonstrate the effectiveness and efficiency of this paradigm. Beyond CTA, D2R also suggests a general task-triggered representation paradigm for table understanding, as discussed in Appendix A.3.

Limitations

D2R shows that a description-triggered decoder state can serve as an effective column representation for CTA, but our evaluation remains within closed-set CTA benchmarks. This focus enables controlled comparison, while leaving a natural opportunity to test whether the same representation benefits related table tasks such as schema linking, table retrieval. More broadly, the paradigm of extracting the hidden state at the description trigger token could be applied to other embedding-based scenarios beyond tables. Moreover, while larger decoder backbones generally improve accuracy (e.g., Qwen3-8B vs. 0.6B), they also increase computational cost. Our default model uses Qwen3-0.6B with LoRA to balance performance and efficiency, but the trade-off between model size, inference speed, and accuracy remains an application-specific decision. Further exploration of efficient lightweight architectures or adaptive early-stopping strategies could address this limitation. Additionally, the optimal prefix width varies across datasets due to schema width and column co-occurrence patterns. We currently select it manually, while future work could filter relevant columns via attention scores or other relevance signals.

Acknowledgments

This work was supported by the Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (JYB2025XDXM103) and the Leading Talent of Technological Innovation Program (No. 2023R5214) of Zhejiang Province.

References

- Muhammad Imam Luthfi Balaka, David Alexander, Qiming Wang, Yue Gong, Adila Krisnadhi, and Raul Castro Fernandez. 2025. [Pneuma: Leveraging llms for tabular data representation and retrieval in an end-to-end system](#). *Proc. ACM Manag. Data*, 3(3).
- Lingxi Cui, Huan Li, Ke Chen, Lidan Shou, and Gang Chen. 2026. [Tabular data augmentation for machine learning: Progress and prospects of embracing generative ai](#). *ACM Comput. Surv.*, 58(12).
- Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. [Turl: table understanding through representation learning](#). *Proc. VLDB Endow.*, 14(3):307–319.
- Zhihao Ding, Yongkang Sun, and Jieming Shi. 2025. [Retrieve-and-verify: A table context selection framework for accurate column annotations](#). *Preprint*, arXiv:2508.17203.
- Lun Du, Fei Gao, Xu Chen, Ran Jia, Junshan Wang, Jiang Zhang, Shi Han, and Dongmei Zhang. 2021. [Tabularnet: A neural network architecture for understanding semantic structures of tabular data](#). In *Proceedings of the 27th ACM SIGKDD Conference, KDD '21*, page 322–331.
- Xingyu Du, Gongsheng Yuan, Sai Wu, Gang Chen, and Peng Lu. 2024. [In situ neural relational schema matcher](#). In *Proc. IEEE ICDE*, pages 138–150.
- Grace Fan, Jin Wang, Yuliang Li, and Renée J. Miller. 2023a. [Table discovery in data lakes: State-of-the-art and future directions](#). In *Companion of the 2023 International Conference on Management of Data*, page 69–75.
- Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée J. Miller. 2023b. [Semantics-aware dataset discovery from data lakes with contextualized column-based representation learning](#). *Proc. VLDB Endow.*, 16(7):1726–1739.
- Benjamin Feuer, Yurong Liu, Chinmay Hegde, and Juliana Freire. 2024. [Archetype: A novel framework for open-source column type annotation using large language models](#). *Proc. VLDB Endow.*, 17(9):2279–2292.
- Simon Gottschalk and Elena Demidova. 2022. [Tab2kg: Semantic table interpretation with lightweight semantic profiles](#). *Semantic Web*, 13(3):571–597.
- Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. [Lora: Low-rank adaptation of large language models](#). *Preprint*, arXiv:2106.09685.
- Madelon Hulsebos, Çagatay Demiralp, and Paul Groth. 2023. [Gittables: A large-scale corpus of relational tables](#). *Proc. ACM Manag. Data*, 1(1).
- Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César Hidalgo. 2019. [Sherlock: A deep learning approach to semantic data type detection](#). In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, page 1500–1508.
- Aamod Khatiwada, Grace Fan, Roei Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J. Miller, and Mirek Riedewald. 2023. [Santos: Relationship-based semantic table union search](#). *Proc. ACM Manag. Data*, 1(1).
- Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. 2020. Supervised contrastive learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS '20*.

- Keti Korini and Christian Bizer. 2023. [Column type annotation using chatgpt](#). *Preprint*, arXiv:2306.00745.
- Keti Korini, Ralph Peeters, and Christian Bizer. 2022. Sotab: The wdc schema. org table annotation benchmark. In *CEUR Workshop Proceedings*, volume 3320, pages 14–19.
- Benno Kruit, Hongyu He, and Jacopo Urbani. 2020. [Tab2know: Building a knowledge base from tables in scientific papers](#). In *International Semantic Web Conference*, page 349–365.
- Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. 2010. [Annotating and searching web tables using entities, types and relationships](#). *Proc. VLDB Endow.*, 3(1–2):1338–1347.
- Ailiang Lin, Zhuoyun Li, Yusong Wang, Kotaro Funakoshi, and Manabu Okumura. 2026. [Causal2vec: Improving decoder-only llms as embedding models through a contextual token](#). In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, page 22769–22788.
- Zhengjie Miao and Jin Wang. 2023. [Watchog: A light-weight contrastive learning based framework for column annotation](#). *Proc. ACM Manag. Data*, 1(4).
- Aniket Pramanick and Indrajit Bhattacharya. 2021. [Joint learning of representations for web-tables, entities and types using graph convolutional network](#). In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 1197–1206.
- Timo Schick and Hinrich Schütze. 2021. [Exploiting cloze-questions for few-shot text classification and natural language inference](#). In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 255–269.
- Oscar Skean, Md Rifat Arefin, Dan Zhao, Niket Nikul Patel, Jalal Naghiyev, Yann LeCun, and Ravid Shwartz-Ziv. 2025. [Layer by layer: Uncovering hidden representations in language models](#). In *Forty-second International Conference on Machine Learning*.
- Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. 2022. [Annotating columns with pre-trained language models](#). In *Proceedings of the 2022 International Conference on Management of Data, SIGMOD ’22*, page 1493–1503.
- Yushi Sun, Hao Xin, and Lei Chen. 2023. [Reca: Related tables enhanced column semantic type annotation framework](#). *Proc. VLDB Endow.*, 16(6):1319–1331.
- Petros Venetis, Alon Halevy, Jayant Madhavan, Marius Pasca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. 2011. [Recovering semantics of tables on the web](#). *Proc. VLDB Endow.*, 4(9):528–538.
- Elena Voita, Rico Sennrich, and Ivan Titov. 2019. [The bottom-up evolution of representations in the transformer: A study with machine translation and language modeling objectives](#). *Preprint*, arXiv:1909.01380.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. [Qwen3 technical report](#). *Preprint*, arXiv:2505.09388.
- An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 43 others. 2024. [Qwen2 technical report](#). *Preprint*, arXiv:2407.10671.
- Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. [TaBERT: Pretraining for joint understanding of textual and tabular data](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8413–8426.
- Dan Zhang, Madelon Hulsebos, Yoshihiko Suhara, Çağatay Demiralp, Jinfeng Li, and Wang-Chiew Tan. 2020. [Sato: contextual semantic type detection in tables](#). *Proc. VLDB Endow.*, 13(12):1835–1848.
- Haochen Zhang, Yuyang Dong, Chuan Xiao, and Masafumi Oyamada. 2024a. [Jellyfish: Instruction-tuning local large language models for data preprocessing](#). In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 8754–8782.
- Haoxiang Zhang, Yurong Liu, Aécio Santos, Wei-Lun Hung, and Juliana Freire. 2025. [Autoddg: Automated dataset description generation using large language models](#). *Preprint*, arXiv:2502.01050.
- Tianshu Zhang, Xiang Yue, Yifei Li, and Huan Sun. 2024b. [TableLlama: Towards open large generalist models for tables](#). In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 6024–6044.

A Appendix

A.1 Additional Experimental Results

Layer sensitivity. The most important sensitivity study is the choice of extraction layer. Figure 8 shows that upper-intermediate layers provide stronger representations than the final decoder layer. This pattern is consistent with the role of decoder layers: the last layer is closest to next-token prediction, while intermediate layers retain more stable semantic information for classification. The performance of $L - 4$ and $L - 8$ is close across the two datasets, so the main experiments average these two trigger states to reduce layer-specific variance.

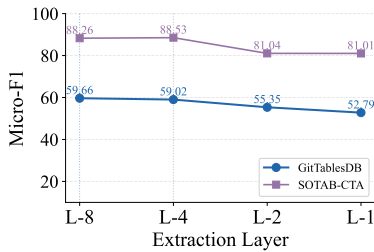


Figure 8: Sensitivity to the extraction layer. The upper-intermediate layers perform best overall, with $L - 4$ and $L - 8$ close.

Classifier head on a smaller backbone. To validate the classifier-head design independently of backbone depth, we add a smaller and shallower backbone Qwen2.5-0.5B (24 layers, about 0.49B parameters), which comes from the same family as our default Qwen3-0.6B (28 layers) and is therefore directly comparable. Under the same prompt format, extraction rule, and classifier, we compare the plain “2-layer MLP” head against the “LN+2-layer MLP” head on this smaller decoder. The results are reported below, alongside the original Table 4 numbers on Qwen3-0.6B.

Head variant	Qwen3-0.6B		Qwen2.5-0.5B	
	GitTablesDB	SOTAB-CTA	GitTablesDB	SOTAB-CTA
2-layer MLP	23.10 / 0.38	1.76 / 0.07	20.53 / 1.29	2.20 / 0.09
LN+2-layer MLP w/o CL	56.76 / 32.96	84.12 / 82.28	56.29 / 31.52	79.35 / 74.79
LN+2-layer MLP (D2R)	60.77 / 36.76	88.79 / 87.67	59.01 / 35.46	83.29 / 78.71

Table 8: Classification-head ablation on Qwen3-0.6B and Qwen2.5-0.5B. Results are Micro-F1 / Macro-F1.

The pattern observed on Qwen3-0.6B holds on the smaller Qwen2.5-0.5B. A plain 2-layer MLP head remains close to collapse on both backbones, while adding layer normalization recovers most of the performance, lifting GitTablesDB Micro-F1 from 20.53 to 56.29 and SOTAB-CTA Micro-

F1 from 2.20 to 79.35 on Qwen2.5-0.5B. Without layer normalization, the raw decoder state has uneven feature scales, which makes optimization unstable. In contrast, layer normalization stabilizes the input distribution so that the head becomes trainable. Contrastive learning then adds a further gain on top of the normalized head, reaching 59.01 and 83.29 respectively. The size of the improvement from layer normalization is therefore not an artifact of a particular backbone depth, and the head design remains effective on a shallower 24-layer decoder.

Effect of sampled values per column. For the rows, for each column we take the first few non-empty values in row order, using a fixed number rather than tuning it per table. To study the effect of the amount of per-column content directly, we add a row-count sweep on GitTablesDB and SOTAB-CTA, comparing our default of 5 values per column against 10 values per column. The results are reported in Table 9.

Values per column	GitTablesDB	SOTAB-CTA
5 (default)	60.77 / 36.76	88.79 / 87.67
10	58.86 / 34.86	87.51 / 86.99

Table 9: Effect of the number of sampled values per column. Results are Micro-F1 / Macro-F1 (%).

Increasing the number of sampled values from 5 to 10 does not improve performance and slightly lowers it on both benchmarks. A likely reason is that including more values per column lengthens the context and can introduce noise rather than useful signal, so a fixed and moderate number of values is a reasonable default. A more adaptive selection of columns and rows is a promising direction, which we leave for future work.

A.2 Discussion on the Decoder-Only Design

D2R uses a decoder-only backbone, but this does not mean that the target-column representation lacks cross-column context. In D2R, the table context is first prefilling into a shared prefix KV cache, and each target column is then processed against this cache. Since the target-column suffix and the description trigger are placed after the table-context prefix, the trigger position can attend to all columns included in the cached prefix under causal self-attention. Across stacked decoder layers, this allows the trigger representation to recursively integrate key-value states from the table context. Therefore, although the attention pattern

is causal rather than bidirectional, the final target-column representation is still conditioned on the surrounding columns. The attention analysis in Figure 5 provides qualitative evidence that the trigger indeed attends to semantically related columns, suggesting that the cached prefix is actively used when forming the column representation.

The decoder-only design is also motivated by the semantic role of the description-triggered state. At the point where the model is prompted to describe the target column, the decoder has already aggregated the target-column values and their table context into a state that is prepared for semantic verbalization. D2R uses this continuous state directly as the column representation, instead of generating a textual description and encoding it with a separate model. This design is consistent with prior observations that intermediate decoder layers often contain strong semantic representations, while the final layer is more directly tied to next-token prediction. In our CTA setting, description-based methods such as Qwen-DescEmbed already show that decoder LLMs can provide useful column semantics, outperforming several encoder-only baselines in Table 1. D2R further exploits this semantic capacity by extracting the description-triggered hidden state and aligning it directly with the CTA objective.

This design also has practical advantages in scalability and deployment. Compared with BERT-style encoder-only models, decoder-only LLMs have become the dominant direction of language-model development and continue to improve in scale, context length, and semantic capacity. Since D2R applies a fixed hidden-state extraction rule to an off-the-shelf decoder, stronger decoder backbones can be incorporated without changing the overall framework. The backbone comparison in Section 4.3 shows that D2R remains effective across different decoder families and model sizes. In addition, D2R requires only lightweight LoRA adaptation together with a small classification head, rather than extensive task-specific pretraining. The longer context windows commonly supported by decoder LLMs are also useful for wide tables, where more surrounding columns may provide disambiguating evidence.

A.3 Extensibility to Other Table Understanding Tasks

The core of D2R is task agnostic, since the representation is obtained by reading the last-token hid-

den state of a description trigger at an intermediate decoder layer. The backbone and the same prefix caching mechanism are reusable across tasks. The task-specific parts are the trigger, the supervision objective, and the head. We discuss schema matching as a representative extension, since it differs from column type annotation in task form, moving from single-column classification to matching columns across two schemas.

In schema matching, the goal is to identify columns from two schemas that correspond to the same real-world attribute. The trigger changes from describing the semantic type of a column to eliciting the real-world attribute that the column most likely represents, so that the decoder encodes each column in terms of the attribute it corresponds to rather than its type label. The supervision objective changes from a classification loss over type labels to a contrastive objective defined over column pairs, where columns that correspond to the same real-world attribute across the two schemas form positive pairs and all other combinations form negatives, so that columns sharing an attribute are aligned in the representation space and unrelated ones are separated. The head then reduces to a similarity between two column representations, and a matching decision is made by comparing this similarity against a threshold, which removes the need for a task-specific classification head.

This example illustrates how D2R can adapt from column type annotation to schema matching. The description trigger mechanism and the backbone are carried over, and the adaptation mainly redefines the supervision objective and replaces the type classifier with a similarity threshold. A full empirical study of transferring D2R to other table understanding tasks such as schema matching is left as future work.

A.4 Prompt Template

In practice, the model is given an instruction that defines the CTA task, followed by serialized column content and a target-column query.

```
[Prefix]
System: You are a semantic type
annotator for tabular data. Analyze
each column's semantic meaning, data
type, value patterns, cross-column
relationships, and domain-specific
interpretation.
User: Table name: <table_id>
Columns:
- col_0: <serialized column text>
- col_1: <serialized column text>
```

Benchmark	# Tables	# Types	Total # Cols	Labeled Cols	Min/Max/Avg
GitTablesDB	3,737	101	45,304	5,433	1 / 193 / 12.1
GitTablesSC	2,853	53	34,148	3,863	1 / 150 / 12.0
SOTAB-CTA	24,275	91	195,543	64,884	3 / 30 / 8.1
WikiTables-CTA	406,706	255	2,393,027	654,670	1 / 99 / 5.9

Table 10: Dataset statistics.

```

...
- col_n: <serialized column text>

[Suffix]
Column "col_i" contains values:
<serialized column text>. Based on
the table above, the semantic type and
meaning of this column is

```

The implementation serializes each column as a line of the form `- col_i: <data>`, and uses the raw column text after truncation. The column representation is then computed from selected decoder layers by pooling token states, with last-token pooling as the default. The released code contains minor prompt-construction variants for efficiency, but they preserve this same interface and target-column query.

A.5 Implementation and Hyperparameters

Hyperparameter	Main setting
Backbone	Qwen3-0.6B
Representation layers	$L - 4$ and $L - 8$ (averaged)
Pooling rule	last-token pooling
L2 normalization	enabled
Classifier head	LN-MLP
Prefix context width	5
Gradient accumulation steps	16
Column mini-batch size	16
Epochs	30
Early-stopping patience	8
Learning rate	1×10^{-4}
Weight decay	0.05
Warmup ratio	0.15
Contrastive weight λ	0.2
Temperature τ	0.07
Optimizer	AdamW with OneCycleLR
Maximum prefix length	256 tokens
Maximum suffix length	512 tokens
Column text truncation	128 characters
LoRA rank / alpha / dropout	32 / 64 / 0.3

Table 11: Main hyperparameters used by the current Qwen3-0.6B experiments and the shared experiment launcher. Some settings are adjusted slightly across ablations and backbone comparisons.

The default D2R uses Qwen3-0.6B with LoRA (rank 32) (Hu et al., 2021). LoRA is applied to all projection matrices, and the classifier head is trained from scratch. For each table, the prefix

is forwarded once under stop-gradient to build a reusable KV cache. All column suffixes are then processed in a single batched forward pass reusing this cache. Gradients flow through the suffix tokens, LoRA adapters, and task heads. The same cache sharing is used at inference. The column embedding is obtained by averaging the last-token hidden states from layers $L - 4$ and $L - 8$, followed by L2 normalization. We set the contrastive weight $\lambda = 0.2$ and temperature $\tau = 0.07$. Prompts and hyperparameters are listed in Appendix A.4 and A.5. Experiments run on a 48-core Intel Xeon Platinum 8476C (440 GB RAM) and two NVIDIA H800 GPUs (160 GB VRAM).

Unless otherwise stated, we use the configuration in Table 11 as the default training setup. Specific ablations and cross-backbone experiments make small adjustments to settings such as representation extraction, context width, LoRA capacity, or mini-batch size when needed.

A.6 Dataset Details

We evaluate on four CTA benchmarks, summarized in Table 10: GitTablesDB, GitTablesSC (Hulsebos et al., 2023), SOTAB-CTA (Korini et al., 2022), and WikiTables-CTA (Deng et al., 2020). GitTablesDB and GitTablesSC are sourced from GitHub and contain real-world relational tables. GitTablesDB annotates columns with DBpedia properties, while GitTablesSC uses Schema.org properties. Both datasets feature wide tables (e.g., GitTablesDB: max 193, avg 12.1 columns), making them suitable for testing cross-column dependencies. SOTAB-CTA focuses on heterogeneous web tables with Schema.org labels and provides a long-tail setting. WikiTables-CTA is the largest benchmark with 406,706 tables, derived by TURL (Deng et al., 2020) from the WikiTables corpus and annotated with 255 DBpedia types.

A.7 Notation

Table 12 summarizes the main symbols used throughout the paper.

Symbol	Meaning	Where used	Symbol	Meaning	Where used
T	Input table	CTA formulation	$\mathbf{K}_{i,\leq t}^{(\ell)}, \mathbf{V}_{i,\leq t}^{(\ell)}$	Suffix keys and values up to token t	Eq. 6
c_i	Target column indexed by i	CTA formulation	$\mathbf{o}_{i,t}^{(\ell)}$	Attention output for suffix token t at layer ℓ	Eq. 6
$y_i, \hat{y}_i$	Gold label and predicted distribution	Classification	d_k	Key/query dimension for attention scaling	Eq. 6
$\mathbf{h}_i$	Column embedding from decoder state	Eqs. 5, 8	$\mathcal{P}_j$	Prefix-token positions of source column c_j	Eq. 7
$\text{GetEmbedding}(T, c_i)$	Table-column embedding function	Overview	$\alpha_{i,t_i,p}^{(\ell)}$	Attention weight from trigger token t_i to prefix position p	Eq. 7
$\text{CLS}(\cdot)$	Classifier over a column embedding	Overview	$a_{i \rightarrow j}^{(\ell)}$	Attention mass from target column c_i to source column c_j	Eq. 7
$\mathcal{M}_\theta$	Decoder-only language model	suffix forward pass	t_i	Final description-trigger token position	Eq. 4
L, ℓ	Number of layers and layer index	Eqs. 2–5	$\mathbf{x}_{i,t}^{(\ell)}$	Decoder hidden state for suffix token t	Eq. 4
$\mathbf{r}_T$	Table-level prefix prompt	Eq. 2	$\ell^* = L - \delta$	Intermediate extraction layer	Eq. 5
$\mathbf{u}_i$	Target-column suffix prompt	Eq. 4	ε	Stability constant for normalization	Eq. 5
$\rho(i)$	Column identifier for c_i	Suffix construction	R	Number of MLP layers	Eq. 8
$\eta(c_i)$	Serialized sampled values from c_i	Suffix construction	$\mathbf{r}^{(q)}$	Representation after the q -th MLP layer	Eq. 8
γ	Description trigger	Suffix construction	$\mathbf{W}^{(q)}, \mathbf{b}^{(q)}$	Parameters of the q -th MLP layer	Eq. 8
$\oplus$	String concatenation operator	Suffix construction	$\mathbf{W}_{\text{out}}, \mathbf{b}_{\text{out}}$	Output-layer parameters	Eq. 8
$\mathcal{C}_T, \bar{\mathcal{C}}_T$	Prefix KV cache and stop-gradient cache	Eqs. 2, 4	$\mathcal{B}$	Mini-batch of target columns	Eqs. 9, 10
$\mathbf{K}_T^{(\ell)}, \mathbf{V}_T^{(\ell)}$	Prefix keys and values at layer ℓ	Eq. 2	$\mathcal{L}_{\text{cls}}, \mathcal{L}_{\text{con}}, \mathcal{L}$	Classification, contrastive, and final losses	Eqs. 9, 10
$\mathbf{q}_{i,t}^{(\ell)}$	Query vector of suffix token t	Eq. 6	$\mathbf{z}_i$	Normalized projection for contrastive learning	Eq. 10
			$P(i), \mathcal{T}, \tau, \lambda$	Positive set, valid anchors, temperature, and contrastive weight	Eq. 10

Table 12: Notation used in the methodology.

A.8 Additional Related Work

Extended CTA background. CTA has evolved from knowledge-base matching and hand-crafted features (Venetis et al., 2011; Limaye et al., 2010) to supervised representation learning over large table corpora (Hulsebos et al., 2019). Early neural models such as SATO (Zhang et al., 2020) exploit neighboring-column context, while most modern CTA systems follow a serialization-then-encoding pipeline: they serialize a target column or an entire table into a text sequence and feed it to a pre-trained encoder, often further adapted with supervised fine-tuning (TURL (Deng et al., 2020)), contrastive pretraining (Starmie (Fan et al., 2023b) and Watchog (Miao and Wang, 2023)), or stronger context retrieval (REVEAL (Ding et al., 2025)). Some work uses LLMs as direct predictors for semantic types (Korini and Bizer, 2023; Feuer et al., 2024), but these methods suffer from high variance, rely heavily on the model’s exposure to table corpora, and incur substantial token costs. A more relevant LLM-assisted CTA paradigm instead leverages an LLM to generate a natural language description of the target column and then encodes that description with a PLM-based encoder (Ding et al., 2025; Balaka et al., 2025), injecting richer column semantics into the representation stage. D2R is most closely related to this latter line, but differs in where the representation is taken: instead of re-encoding a generated description, it uses the decoder state that initiates the description.

Table representation and structural modeling.

The broader table understanding literature has proposed several ways to encode rows, columns, and local structure. TaBERT jointly pretrains over textual and tabular inputs (Yin et al., 2020),

while TURL adapts transformer pretraining objectives to relational tables and entity-aware structure (Deng et al., 2020). Graph-based table encoders offer another way to preserve structural relations: TabularNet (Du et al., 2021) builds graph-structured representations of cells and table semantics, and TabGCN (Pramanick and Bhattacharya, 2021) jointly learns representations for web tables, entities, and types with graph convolutional networks. Together, these works show that column meaning is shaped by both local value patterns and table-level context, although the resulting representations are still mainly obtained from encoder-side structural modeling rather than decoder states prepared for semantic verbalization.

A.9 Ethics Statement

We hereby confirm that the entire work presented in this paper, including all content, analysis, and conclusions, is the original work of the human authors. While the core research, analysis, and conclusions are entirely the work of the human authors, generative AI was consulted for the limited purpose of improving the grammar and readability of the text.