TASTE: Towards Practical Deep Learning-based Approaches for Semantic Type Detection in the Cloud

Tao Li*†
Database Group
State Cloud, China Telecom
Shenzhen, China
lit51@chinatelecom.cn

Feng Liang*‡
AI Research Institute
Shenzhen MSU-BIT University
Shenzhen, China
fliang@smbu.edu.cn

Jinqi Quan
Database Group
State Cloud, China Telecom
Shenzhen, China
quanjq1@chinatelecom.cn

Chuang Huang
Database Group
State Cloud, China Telecom
Shenzhen, China
huangc41@chinatelecom.cn

Teng Wang
Database Group
State Cloud, China Telecom
Guangzhou, China
wangt_5@chinatelecom.cn

Runhuai Huang
Intelligent Edge Department
State Cloud, China Telecom
Guangzhou, China
huangrh@chinatelecom.cn

Jie Wu
Cloud Computing Research Institute
China Telecom
Beijing, China
wujie@chinatelecom.cn

Xiping Hu†‡
AI Research Institute
Shenzhen MSU-BIT University
Shenzhen, China
huxp@smbu.edu.cn

ABSTRACT

In recent years, we have witnessed more and more data management, preparation, and wrangling services appearing in the cloud. Semantic type detection is important for these services that rely on semantic types to interpret data and provide useful functions accordingly. Meanwhile, deep learning (DL) has been introduced for semantic type detection and transforming the field. However, existing DL-based approaches, albeit successfully achieving high F1 scores, are not practical in the real cloud environment because they suffer from issues like low efficiency and high intrusiveness to user data sources.

To address these issues, we present TASTE, a novel semantic type detection framework with two phases, each associated with a DL-driven detection task. The intuition behind this framework is that metadata (e.g., column name, statistics) contain rich technical and business information, which can be leveraged to detect semantic types effectively while only incurring lightweight impact on user data sources. Thus, we design the detection task in the first phase purely using native metadata from user data sources as input. In contrast, the second phase is optional and only activated when there is a high uncertainty with the first task's result. It then needs to retrieve both metadata and column content to derive semantic types more reliably. Furthermore, we adopt multi-task learning and develop a novel DL model, called *Asymmetric Double-Tower Detection (ADTD)*, to support the two tasks simultaneously. This design enables caching and reuse of the latent representations from the first task to reduce inference time. In the implementation, we further introduce a pipelined execution mechanism to boost performance for massive user table processing. Evaluation results show that TASTE achieves state-of-the-art performance in execution time and F1 score, and is more robust under different data privacy settings, demonstrating its potential for real application in cloud environment. [1]

1 INTRODUCTION

The importance of semantic type detection has been well recognized in the domain of data management [5, 18], data preparation [3, 19], data wrangling [23] and web table understanding [12]. *Semantic types* uncover the semantic meaning of data and usually map to real-world concepts or entities. Thus, they can help human understand data, and more importantly, can be utilized by data management and preparation systems to provide proper functions (like searching, transformation, and cleaning) or automate certain tasks. For example, data with semantic type "credit card number" can be automatically identified as sensitive by data protection systems, which further can provide data masking functions.

Relying on human efforts to detect semantic types is obviously impractical in the big data era. Instead, we resort to various algorithms, from simple regular expressions to complex machine learning models, to achieve *automatic semantic type detection*. In recent years, many deep learning-based (DL-based) approaches have been proposed to detect semantic types, such as [14, 17, 19, 30]. They have shown great potential and established new state-of-the-art performance (in terms of F1 score). For example, Sherlock [19] extracts 1,588 features from column content as the input for a deep neural network. It achieves an F1 score of 0.89 and outperforms a wide range of traditional methods, including dictionary lookup, regular expression matching and decision tree. Subsequent work, such as TURL [17] and DODUO [30], have raised an F1 score to another level by further incorporating pre-trained language models.

Meanwhile, as the cloud computing paradigm has become mature, traditional on-premise data management and preparation

*These authors contributed equally to this work, ordered alphabetically by surname.
†Corresponding authors.
‡Also affiliated with Guangdong-Hong Kong-Macao Joint Laboratory for Emotional Intelligence and Pervasive Computing, Shenzhen MSU-BIT University, China

[1]The source code is available at https://github.com/ncols-bytes/taste.git

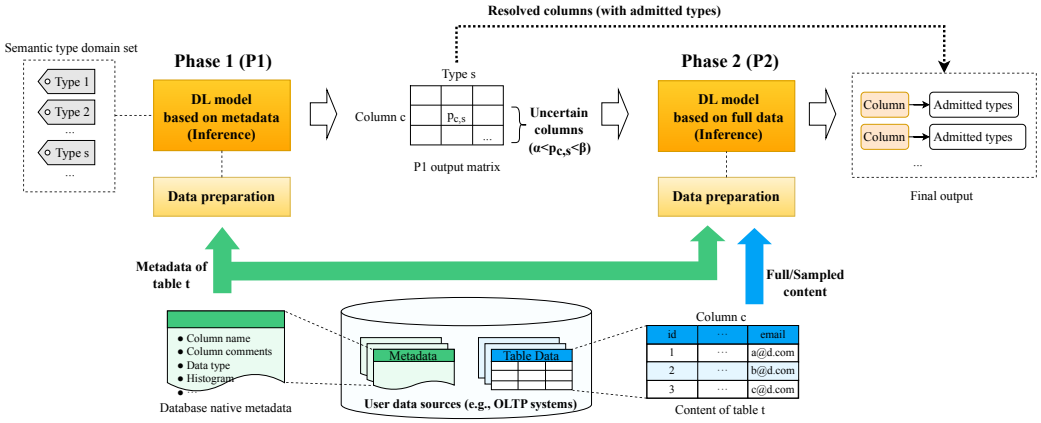


Figure 1: Overview of the Taste framework.

systems have been migrating to the cloud. Major cloud service providers are also providing similar services, such as Microsoft Purview [5], AWS Glue Data Catalog [2], Google Looker Studio [3], and Alibaba Cloud Data Security Center [1]. Unfortunately, existing DL-based approaches like [14, 17, 19, 30], when applied in the cloud context, can suffer from two major drawbacks. First, they are intrusive to user data sources and thus may not be acceptable to users. These DL models rely on features extracted from column content, and thus they must scan user data sources to retrieve data. For example, the model in [30] needs to get all column content and concatenate them as model input. On one hand, scanning column content causes increased I/O and connections on user data sources (e.g., OLTP systems), which can potentially disrupt their business. On the other hand, users with strong concerns about data leakage, auditing, and compliance tend to disallow cloud services to access and examine their column content [9]. Second, the existing approaches are not execution efficient. Column scanning operations are costly and abundant column features can increase the complexity of the DL model. Moreover, these approaches generally run in sequential execution mode, which processes batches of tables one by one without utilizing cross-batch concurrency. Considering the large number (as high as billions) of tables and columns from diverse tenants to deal with and the high detection frequency in the cloud, efficiency is a primary concern to cloud service providers. Even small savings in time for one column can collectively bring significant benefits. In this study, we try to explore an efficient DL-based semantic type detection solution that can be practically employed in cloud services, especially for relational data (tables) which are major enterprise data forms in the cloud.

To circumvent the above drawbacks, we develop a two-phase semantic type detection framework, called Taste. The execution flow of the Taste framework is depicted in Fig. 1. We divide the traditional one-shot prediction into two phases, each of which runs a separate DL-based detection task. The first phase (i.e., P1), purely leverages native metadata from the data sources to predict semantic types, whereas the second phase (i.e., P2) utilizes full table information, including both metadata and column content. This design is motivated by the observation that metadata commonly contains rich information in the technical level (e.g., data types), business level (e.g., comments and names), and content level (e.g., histogram), which can be exploited to infer correct semantic types. For example, cloud tenants tend to use meaningful table/column names and comments, and it is common for enterprises to enforce various standards for defining table schema. Meanwhile, retrieving metadata is a much less costly and intrusive operation compared to data scan. Therefore, we can quickly get a preliminary understanding about the columns through a metadata-based model in P1. Based on the output of P1, we are able to assess the relevance between each semantic type and each column. P2 is on-demand, and only activated if we need to involve more data to verify uncertain semantic types. For example, for a column with the name "num" without additional comments, P1 is uncertain that it belongs to type "phone number" or "credit card number". It is necessary to launch P2 to check the column content to verify. Through the combination of P1 and P2, Taste can not only run efficiently and mitigate the negative impact on user data sources, but also achieve robust semantic type detection when metadata quality is not satisfactory. Furthermore, the Taste framework is flexible in the sense that cloud tenants concerned about data exposure to cloud service providers can choose to disable P2 completely.

To fit Taste's two-phase execution framework, we further design a novel *Asymmetric Double-Tower Detection (ADTD)* model by incorporating multi-task learning for the two phases. This asymmetric Transformer-based architecture has been widely adopted in modern large language models (LLMs) including GPT-3 and its variants and competitors [11, 15, 28], and we adapt it to facilitate the execution of the two-phase framework. ADTD consists of two towers of multiple layers of Transformer blocks, named *metadata tower* and *content tower*, respectively. These two towers can be considered as two expert submodels that are responsible for encoding latent representations of metadata and column content, respectively. The final prediction of ADTD is based on the combination of the two towers, using the automatic weighted loss for multi-task learning as the objective. During inference, the metadata tower is extracted from the ADTD model and serves as the model for P1, while the whole ADTD model with both towers is used to serve P2. In this way, the ADTD model is trained once but applied for different tasks in P1 and P2. The asymmetry of ADTD's structure lies in that the content tower relies on the metadata tower, but not vice versa. The input of every layer of content tower requires not only content latent representations, but also metadata latent representations from the metadata tower. Exploiting this asymmetric dependency, we build *latent caches* in the metadata tower to store metadata latent representations, which can be reused by P2. This thus avoids duplicate computation of these values and improves inference

efficiency. For capturing textual and tabular context in the Transformer blocks of the two towers, they share the same parameters and are pre-trained on a table corpus before they are later trained for the semantic type detection task.

Moreover, we develop an efficient implementation of the Taste framework by using *pipelining*. We divide each phase of Taste into two stages: data preparation (S1) and semantic type inference (S2). These stages of different phases can be parallelized because they primarily require different types of resources: S1 uses I/O and CPU, and S2 uses GPU. Therefore, when one detection job enters the S2 stage, another can be activated to enter the S1 stage, creating an execution pipeline that can process multiple jobs at the same time. By fully utilizing all types of resources, pipelining significantly enhances the overall execution efficiency, and enables the processing of a large number of tables and columns in the cloud.

Finally, we evaluate Taste on two popular datasets for semantic type detection, WikiTable [10] and GitTables [20], and compare its performance with existing approaches, i.e., TURL [17] and Doduo [30]. Evaluation results show that Taste achieves better F1 scores of 0.9340 and 0.9909 on WikiTable and GitTables, respectively. Most importantly, Taste can reduce execution time and the ratio of scanned columns by up to 85.0% and 99.1%, respectively. The performance gains become even more pronounced when the ratio of columns without any semantic type grows, which is pervasive in the real cloud environment. Moreover, compared to existing approaches, Taste is more robust under different data privacy settings, and thus more practical in real cloud environments.

Our key contributions are summarized as follows.

- We propose Taste, a high-performance and flexible semantic type detection solution that is suitable for cloud services that need to address scalability and user acceptance issues.
- We devise a novel ADTD model based on multi-task learning with latent cache and a pipelined execution algorithm to improve the performance of the Taste framework.
- Evaluation results based on open datasets have shown that Taste establishes state-of-the-art performance in terms of the execution time and F1 score while causing significantly lower intrusion into user data sources.

The rest of the paper is organized as follows. Section 2 provides some preliminaries about applications and techniques of semantic type detection, as well as the end-to-end semantic type detection problem in the cloud. Section 3 describes the overview and the two-phase design of the Taste framework. Details of the ADTD model are introduced in Section 4 and an efficient implementation of Taste is presented in Section 5. Evaluation results are shown and discussed in Section 6, which is followed by a summary of related work in Section 7. Section 8 concludes this paper and discusses some promising directions for future work.

2 PRELIMINARIES

2.1 Applications of Semantic Types

Different from raw data types, such as integer, float, and string, semantic types are inferred knowledge that can characterize the high-level and domain-specific notions of data. Actually, the usefulness of semantic types has been well acknowledged by academia and industry. So far, they have found applications in a wide range of scenarios, such as table understanding, data cataloging and search [16, 26], data quality validation [29, 34], data transformation, data wrangling [23], etc. Commercial products with semantic type detection capabilities include Alteryx Trifacta [8], Microsoft PowerBI [4], and Tableau [7]. For example, Alteryx Trifacta can recognize 10 complex data types, e.g., social security number and phone number, based on regular expression, and accordingly provide data validation and transformation functions. Recently, we have witnessed major cloud service providers incorporating automatic semantic type detection into their services, such as Microsoft Purview [5], AWS Glue Data Catalog [2], Google Looker Studio [3], and Alibaba Cloud Data Security Center [1]. For example, Microsoft Purview provides a data mapping feature that can automatically discover and assign unique logical tags or classes to the data assets, such as the passport number, driver's license number, or credit card number. Those tags then can help users understand the potential risks, protect sensitive data, and search assets efficiently.

2.2 End-to-end Semantic Type Detection For Relational Data In the Cloud

The importance of semantic types have motivated cloud providers to continuously improve the semantic type detection capabilities in their products. Meanwhile, relational data (i.e., tables) are of their primary focus due to the pervasiveness in the cloud and high value to users. Commonly, relational data exist in user's databases, such as OLTP systems (e.g., MySQL, PostgreSQL) and data warehouse/OLAP systems (e.g., AWS Redshift, Snowflake, SparkSQL). However, when cloud providers (like us) adopt DL-based approaches to detecting semantic types in the cloud context, they face new challenges, such as efficiency, scalability, data exposure, etc. To this end, we introduce the semantic type detection task from cloud service providers' perspective, which we call *end-to-end semantic type detection*: Given the domain set of semantic types S and a table t with metadata and column content stored in remote user databases, the goal of end-to-end semantic type detection is to predict a set of semantic types from S for each column c of table t in an efficient and user-friendly manner. The predicted results can be empty or contain multiple types, forming a multi-label classification problem.

Note that the end-to-end semantic type detection task has multiple objectives. Apart from predicting semantic types accurately, we are also concerned with some practical issues, such as efficiency and user satisfaction, which have not been sufficiently addressed in the existing work, such as [14, 17, 19, 30]. First, we will consider runtime efficiency when designing the semantic type detection mechanisms. Specifically, we propose an end-to-end execution time metric, which involves both data retrieval time from remote data sources and prediction time, thus more realistic. Second, we will endeavor to minimize the impact on user data sources by reducing the number of columns to scan. This enables the solution to meet service-level objectives (SLOs), especially for users who are concerned about I/O isolation, data leakage, and auditing.

2.3 Transformer-based Models for Textual Representation

The Transformer [31] has become the state-of-the-art DL model for learning tasks with sequential data, such as natural language processing (NLP). As illustrated in Fig. 2, the Transformer introduces a *multi-head self-attention* mechanism, which enables sequential data to attend to important parts within the context.

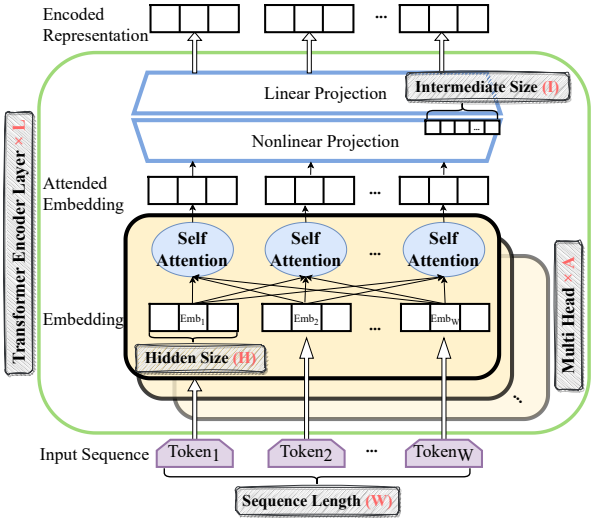


Figure 2: An illustration of Transformer encoder for textual representation

This mechanism is computationally efficient as its heaviest computation is matrix multiplication, which can be parallelized on GPU. Language models, such as BERT [24] and TinyBERT [22], can stack multiple layers of bidirectional Transformer encoders for extracting textual representations. Generally, a BERT-based model can be characterized by five parameters, namely the number of layers (L), the number of self-attention heads (A), the maximum input sequence length (W_{max}), the intermediate size (I) and the hidden size (H). Given the input sequence length W ($W \leq W_{max}$), the model's computational complexity is dominated by $O(LW^2H + LWIH)$. In practice, language models are usually pre-trained on large corpora with tailored language tasks, such as Masked Language Modeling, and later fine-tuned for various downstream tasks. In fact, our method also adopts this pre-training and fine-tuning paradigm.

3 THE TASTE FRAMEWORK

We propose Taste, a two-phase framework for practical end-to-end semantic type detection in the cloud. This section first describes an overview of the Taste framework, and then introduces the constituent two phases in sequence. Table 1 summarizes common notations used throughout this paper.

3.1 Overview

Taste is a *table-wise* two-phase processing framework, as shown in Fig. 1. For a table with multiple columns as the input to Taste, let C denote the set of columns; Taste can output multiple semantic types in S for each column $c \in C$. In this sense, Taste is multi-input and multi-output. The benefit is that table-level metadata can be shared among different columns, and correlations among columns can be captured and utilized for more accurate semantic type detection.

*Definition 3.1 (**Admitted Type**).* If the Taste framework determines that a column $c \in C$ belongs to a semantic type $s \in S$, then we say s is an admitted type to column c.

We can denote the final output of admitted types for column c by A^c. The primary goal of Taste is to find all admitted types for all columns in a time-efficient manner with user-acceptable impact.

Table 1: Notations in the paper.

Notation	Description
S	semantic type domain set
C	column set $\{c\}$
c	a column in C denoted by $(\mathcal{M}^c, \mathcal{D}^c)$
$\mathcal{M}^c$	metadata pertaining to column c, $(\mathcal{M}^c_t, \mathcal{M}^c_n)$
$\mathcal{M}^c_t$	textual metadata pertaining to column c
$\mathcal{M}^c_n$	non-textual metadata pertaining to column c
$\mathcal{D}^c$	column content for column c
$p_{c,s}$	output probability of matching column c to type s by a DL model
C_u	the set of uncertain columns after P1
A^c_1	the set of admitted types for column c after P1
A^c_2	the set of admitted types for column c after P2
A^c	the final output of admitted types for column c

From Fig. 1, we can see Taste consists of two phases, *Phase 1* and *Phase 2*, or P1 and P2 in short, each of which is associated with a semantic type detection task. P1 task is purely metadata-driven and mandatory, whereas P2 task relies on full data, including metadata and column content, and is enabled only when we lack confidence on the detection results of P1.

Each phase further comprises two sequential stages: *data preparation* and *inference*. The data preparation stage is responsible for connecting to user databases, collecting and preprocessing metadata or column content. The inference stage uses DL models to predict semantic types based on the input metadata or column content. We distinguish these two stages with the purpose of increasing parallelism during execution, because these stages have different resource requirements. Specifically, the data preparation stage mainly consumes CPU and I/O resources, while the inference stage is GPU intensive. This yields opportunities for orchestrating them in pipelines, thus improving execution efficiency for multiple tables. We will detail the implementation of the pipelined execution of Taste in Section 5.

3.2 Phase 1

Recognizing the rich information embedded in metadata, in P1 we merely leverage metadata for semantic type detection. This is inspired by the observation that semantic types are closely related to metadata in practice. A trivial scenario is that a semantic type is alphabetically similar to the column name. In fact, users tend to use meaningful names or comments when defining tables/columns, just like we define descriptive names for variables and write comments during coding. In the data preparation stage, various metadata can be fetched readily through API or SQL provided by databases. For example, the `information_schema` database, which contains a variety of schema information, is part of SQL-92 standard. For databases complying with SQL-92 standard, we can run SQL query `SELECT * FROM information_schema.columns` to get column metadata. Also note that the available metadata vary from database to database, and from instance to instance. Some metadata are mandatory, such as table and column names, whereas some are optional, such as table and column comments and histograms. The histogram is a special type of metadata that reflects the characteristics of data distribution. Nevertheless, all these metadata are converted to feature representations, and then fed into our DL model of P1.

In the inference stage, P1 launches a multi-label prediction task using the metadata tower submodel of ADTD. The output is a probability matrix, where each element $p_{c,s}$ represents the probability that semantic type s is related to a column c. Based on these probability values, we then can classify the semantic types for each column. Specifically, we introduce two probability thresholds, α and β, to measure the certainty about the relevance between s and c, where $0 \leq \alpha \leq \beta \leq 1$. If $p_{c,s} \geq \beta$, we are confident that semantic type s is related to column c, and can be accepted as an admitted type to c directly. Let A_1^c denotes the set of admitted semantic types for column c in P1, then we have: $A_1^c = \{s \in S | p_{c,s} \geq \beta\}$.

On the contrary, if $p_{c,s} \leq \alpha$, semantic type s is considered to be irrelevant to column c. Both cases imply that the metadata contains sufficient information to derive the detection results. In the case when $p_{c,s}$ falls into the range (α, β), P1 is uncertain about the detection result. We define the notion of uncertain column as follows.

*Definition 3.2 (**Uncertain Column**).* A column $c \in C$ is called an uncertain column if there exists a semantic type $s \in S$ such that $\alpha < p_{c,s} < \beta$.

Meanwhile, if a semantic type s satisfying $\alpha < p_{c,s} < \beta$, it is called an *uncertain type* for column c. Let C_u denote the set of uncertain columns. If C_u is not empty, P2 is required to further detect semantic types for these uncertain columns based on metadata and column content. Otherwise, P2 can be skipped, and the admitted types inferred by P1 become the final results. Note that one-phase DL models based merely on metadata are a special case of Taste. When α is identical to β, there is no chance that uncertain types can happen, and thus P2 is always skipped. It becomes an ideal option for users who prefer to disallow content examination from cloud services. About the settings of α and β, they can be application-dependent, and cloud users can customize these settings. Our evaluation results in Section 6.7 indicate that by tuning α and β, we can balance the performance, runtime cost, as well as data exposure.

3.3 Phase 2

In contrast to P1, the data preparation stage of P2 additionally fetches the content of all the columns in C_u. Depending on the volume of the column content, either a full scan or sampling of the table can be adopted. For columns in $C \setminus C_u$, we do not fetch their content to save cost and reduce execution time. However, their corresponding metadata are not discarded; instead, the complete information of column-level and table-level metadata are used in order to preserve the cross-column and column-to-table attention in the DL model for P2.

Based on this model, the inference stage will output a set of admitted semantic types, denoted by A_2^c, for each column c in C_u. By combining the results of P1 and P2, the final output of the entire framework is:

$$A^c = \begin{cases} A_1^c & \text{if } c \in C \setminus C_u, \\ A_2^c & \text{if } c \in C_u. \end{cases}$$

Overall, through this two-phase approach, we not only can achieve efficient and low-impact semantic type detection by taking full advantage of metadata, but also guarantee the reliability of the detection results by inspecting data content whenever necessary. In the next section, we will introduce how to build a multi-task learning model to power the Taste framework.

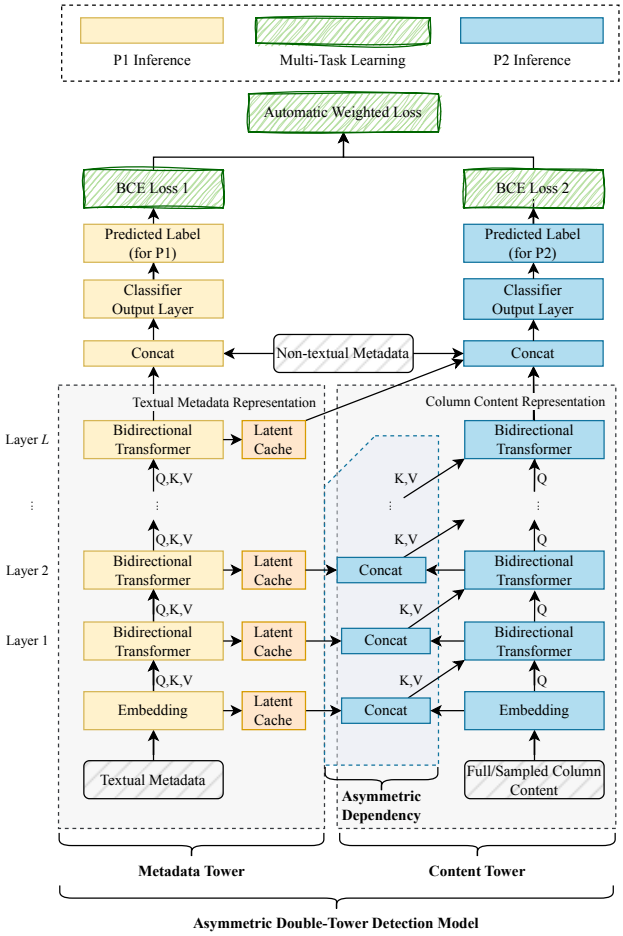


Figure 3: The structure of the Asymmetric Double-Tower Detection (ADTD) model, comprising metadata and content towers with an automatic weighted loss.

4 ASYMMETRIC DOUBLE-TOWER DETECTION MODEL

The Asymmetric Double-Tower Detection (ADTD) model is a multi-task learning network optimized for the two-phase framework. We leverage multi-task learning because the two detection tasks in P1 and P2 are highly related and share metadata as the common input. Basically, the ADTD model is characterized by the following features. First, the model logically consists of two towers with multiple layers of Transformers: *metadata tower* and *content tower*, for encoding latent representations of metadata and column content, respectively. Second, the content tower asymmetrically relies on the intermediate results of the metadata tower to address the attention of metadata and content. Third, these two towers use common Transformer blocks with shared parameters. Fig. 3 illustrates the structure of the ADTD model, where the gray dashed rectangle frames represent the metadata and content towers, respectively, and the blue dashed trapezoid frame denotes the asymmetric dependency.

These two towers can be regarded as two expert submodels for detecting semantic types based on metadata only and based on full table information, respectively. Combining their results has the potential to improve the prediction performance while not undermining the execution efficiency. When ADTD is well

trained, P1 utilizes the metadata tower as the model to infer semantic types based on metadata input. In contrast, P2 utilizes the whole model of ADTD, including both metadata tower and content tower, for inference. The remaining of this section introduces the details about each component of the ADTD model.

4.1 Embedding Layer

The embedding layers of the metadata and content towers use textual metadata and column content as the input, respectively. The embedding layers output fixed-size embeddings as input to the encoder network in the corresponding tower. Given column-related metadata $\mathcal{M}^c$, it can be further divided into textual metadata $\mathcal{M}_t^c$ and non-textual metadata $\mathcal{M}_n^c$. Specifically, $\mathcal{M}_t^c$ includes table name/comments, column name/comments, etc., while $\mathcal{M}_n^c$ is typically categorical or numerical, such as column data type, histogram type (equal-height/equal-width), number of distinct values, etc. The column content, denoted by $\mathcal{D}^c$, is the textual form of sampled or entire cell values concatenated together with a separator, i.e., [CLS]. The embedding output for textual metadata in the metadata tower can be represented by $Embed^{\mathcal{M}_t^c}$, and the one for column content in the content tower can be represented by $Embed^{\mathcal{D}^c}$, respectively. The output dimension of the embedding layer for each token is 312, matching the input dimension of the encoders.

4.2 Encoder Network

4.2.1 Pre-training. Rather than training the encoder network from scratch, we use a pre-trained language model and later fine-tune it for our task. This is currently the best practice for DL-based tasks related to natural language processing (NLP). The encoder network is pre-trained on an unlabeled Wikipedia table corpus [10] with Masked Language Model and Masked Entity Recovery objectives sequentially to capture syntax, semantic, and language and table contextual information [17]. It consists of multi-layer pre-trained bidirectional Transformer blocks, or Transformer blocks in short, with a structure similar to the TinyBERT [22] encoder. The two towers use shared parameters for each layer of Transformers. Besides, the configurations for the five parameters in our model (defined in Section 2.3) are $L = 4$, $A = 12$, $T_{max} = 512$, $I = 1200$ and $H = 312$. With regard to the encoder's input, the sequence contains 150 tokens for table-level metadata, 10 tokens for each column's metadata, and 10 tokens for each cell value.

4.2.2 Encoding in the Metadata Tower. Let $T_i(Q, K, V)$ denote the Transformer block of the i-th layer, where Q, K, and V are the inherent input arguments of query, key, and value, and $Encode_i^{\mathcal{M}_t^c}$ denote the metadata latent representation output by the i-th Transformer, we have:

$$\begin{aligned} Encode_0^{\mathcal{M}_t^c} &= Embed^{\mathcal{M}_t^c} \\ Q_{m_i} = K_{m_i} = V_{m_i} &= Encode_{i-1}^{\mathcal{M}_t^c} \\ Encode_i^{\mathcal{M}_t^c} &= T_i(Q_{m_i}, K_{m_i}, V_{m_i}) \end{aligned}$$

Let $Encode_{meta}(\cdot)$ denote the encoder network of the metadata model, the output of encoding the embedding of the metadata of column c can be represented as:

$$Encode_{meta}(Embed^{\mathcal{M}_t^c}) = Encode_4^{\mathcal{M}_t^c}.$$

Furthermore, thanks to multi-task learning which allows sharing of subnetwork structure, the output of the Transformer blocks in P1's model can be reused by P2's model. To this end, we introduce a cache, called *latent cache*, in the metadata tower which stores the metadata latent representations of each Transformer block, i.e., $Encode_i^{\mathcal{M}_t^c}$.

4.2.3 Encoding in the Content Tower. The encoding in the content tower relies asymmetrically on metadata and column content. Transformer blocks in the content tower share the same parameters as those in the metadata tower. For the Transformer block T_i, we denote the output latent representations of metadata and content by $Encode_i^{\mathcal{M}_t^c}$ and $Encode_i^{\mathcal{D}^c}$, respectively. During inference, the value of each $Encode_i^{\mathcal{M}_t^c}$ can be derived directly from the latent cache, eliminating redundant computation. Regarding the input of T_i in the content tower, both arguments of key K_{c_i} and value V_{c_i} are the concatenation of the metadata and content latent representations of the previous Transformer layer, and the argument of query Q_{c_i} is the content latent representation only. Therefore, we have:

$$\begin{aligned} Encode_0^{\mathcal{D}^c} &= Embed^{\mathcal{D}_c} \\ Q_{c_i} &= Encode_{i-1}^{\mathcal{D}^c} \\ K_{c_i} = V_{c_i} &= Encode_{i-1}^{\mathcal{M}_t^c} \oplus Encode_{i-1}^{\mathcal{D}^c} \\ Encode_i^{\mathcal{D}^c} &= T_i(Q_{c_i}, K_{c_i}, V_{c_i}) \end{aligned}$$

Let $Encode_{cont}(\cdot)$ denote the encoder network of the content tower, the output of encoding the content of column c can be represented as:

$$Encode_{cont}(Embed^{\mathcal{M}_t^c}, Embed^{\mathcal{D}_c}) = Encode_4^{\mathcal{D}^c}.$$

4.3 Single Task Objective

The final latent representations output by both towers are concatenated with other non-textual metadata features as inputs to the corresponding classifier networks, respectively. For simplicity, we name these two classifier networks as the metadata classifier and content classifier, respectively, though the content classifier does not purely depend on column content. Each classifier network is a fully connected feed-forward neural network with a hidden layer activated by ReLU and the output layer activated by sigmoid functions to estimate the probabilities of multiple semantic types. The number of neurons in the hidden layer is 500 and 1000 for the metadata tower and the content tower, respectively. As an example of non-textual metadata, column statistical histogram describes the distribution of the column cell values in different buckets. We can characterize histogram by several features, such as histogram type, number of buckets, and bucket values, etc.

The content classifier also has an asymmetric dependency. Each classifier corresponds to a semantic type detection task with different inputs. Let $Classify_{meta}(\cdot)$ and $Classify_{cont}(\cdot)$ represent the metadata and content classifiers, respectively, the former classifier relies on metadata latent representations only, whereas the latter relies on latent representations of both metadata and column content. Predicting semantic types of column c with the metadata classifier can be represented as:

$$f_1(c) = Classify_{meta}(Encode_4^{\mathcal{M}_t^c} \oplus \mathcal{M}_n^c),$$

where $\oplus$ is the vector concatenation operator. In contrast, predicting semantic types of the column c with the content classifier can be represented as:

$$f_2(c) = Classify_{cont}(Encode_4^{\mathcal{D}^c} \oplus Encode_4^{\mathcal{M}_t^c} \oplus \mathcal{M}_n^c).$$

Each task uses the multi-label binary cross-entropy (BCE) loss as its objective function. For a semantic type $s \in S$ and a column $c \in C$, the ground truth is denoted by $y_{c,s} \in \{0, 1\}$ and the output estimated probability is denoted by $p_{c,s} \in [0, 1]$. Suppose the mini-batch size is b, the multi-label BCE loss is defined as:

$$\mathcal{L}_{BCE}(p, y) = -\frac{1}{b} \sum_{c \in C} \sum_{s \in S} y_{c,s} log(p_{c,s}) + (1 - y_{c,s}) log(1 - p_{c,s})$$

4.4 Multi-Task Learning Objective

The final prediction of ADTD uses the automatic weighted loss as the objective function, based on the multi-task learning of the two towers. Given the losses of the metadata and content towers, denoted by $\mathcal{L}_{BCE_i}, i \in \{1, 2\}$, respectively, the final loss is their weighted sum. Instead of statically assigning values to the weights of these losses, we use learnable parameters as weights to automatically tune the optimal balance between the losses, akin to the method used in [25]. Therefore, the automatic weighted loss is:

$$\mathcal{L}_{ADTD}(\mathcal{L}_{BCE}) = \sum_{i=1}^{2} \frac{1}{2w_i^2} \mathcal{L}_{BCE_i} + ln(1 + w_i^2),$$

where $w \in \mathcal{R}^{2\times 1}$ is a vector of learnable parameters. The w^2 part enforces positive weights for combining the losses of multiple tasks, and the $ln(\cdot)$ part enforces positive regularization values.

5 IMPLEMENTATION

This section describes an efficient implementation of the Taste framework for processing a large number of user tables in the real cloud environment where inter-table parallelism become crucial. Recall that there are two stages for each phase of Taste: data preparation (S1) and inference (S2). Overall, there are four stages for processing a table in the Taste framework. Our implementation is motivated by the observation that different stages have different resource requirements, i.e., the data preparation stage primarily consumes I/O and CPU resources, while the inference stage consumes GPU resources. This is an ideal scenario to leverage *pipelining* to improve runtime efficiency. In the pipelined mode, the processing of different stages for different tables can be interleaved. For example, when one table enters inference stage, the preparation stage of another table can be activated instantly. In contrast, most of the existing work, e.g., [17, 30], adopts a sequential mode to process tables. Apparently, pipelined execution results in higher resource utilization and shorter execution time compared to sequential execution.

In real-world scenarios where modern CPU and GPU are powerful to process multiple tasks simultaneously and there are a number of tables as input, we propose a scheduling algorithm (see Algorithm 1) to orchestrate the stages from different tables by obeying the pipelining logic. The input to the algorithm is a batch of tables to process. Generally, we recommend these tables come from a common database. Then, it can enable us to reuse the database connection, whose creation is generally costly. From line 1-2, we construct two thread pools, $TP1$ and $TP2$, to process data preparation and inference stages, respectively. The sizes of the pools are hardware-dependent. In line 3, we construct a queue Q to hold the stages to process and initialize it to $\emptyset$. From line 4-6, the stages are generated and added into Q. Note that for each table the stages are generated in order, i.e., P1-data preparation, P1-inference, P2-data preparation, and P2-inference. Then, the queue of stages is scheduled to run on the thread pools whenever there are free threads in the pools (lines 7-23). If $TP1$ has an empty slot, we find the first eligible data preparation stage in the queue, denoted as g, and dispatch it to $TP1$ for execution.

Algorithm 1: Scheduling algorithm for pipelined execution of the Taste framework

```
   Input: A batch of tables T to process
1  construct thread pool TP1 for data preparation stages;
2  construct thread pool TP2 for inference stages;
3  initialize the stage queue Q ← ∅;
4  foreach table t ∈ T do
5  |  generate four stages in order for t and add to Q;
6  end
7  while true do
8  |  if TP1 is not full then
9  |  |  g ←poll the first eligible S1 stage from Q;
10 |  |  if g ≠ null then
11 |  |  |  dispatch g to TP1;
12 |  |  end
13 |  end
14 |  if TP2 is not full then
15 |  |  g ←poll the first eligible S2 stage from Q;
16 |  |  if g ≠ null then
17 |  |  |  dispatch g to TP2;
18 |  |  end
19 |  end
20 |  if Q is empty then
21 |  |  break;
22 |  end
23 end
```

*Definition 5.1 (**Eligible Stage**).* A stage is called eligible if its previous stages for the same table have already finished.

Essentially, an eligible stage means the stage is safe to be chosen for execution. Similarly, if $TP2$ is not full, we find the first eligible inference stage in the queue and dispatch it to $TP2$ for execution.

From a single table's point of view, the order of executing its four stages is guaranteed to be correct, because only eligible stages can be dispatched for execution. From concurrency's point of view, multiple tables can be processed in different stages at the same time.

To implement the pipelined version of the Taste framework, we use the Python language, which allows us to easily glue the stage scheduling algorithm and the model inference in a single project. The thread pools in Algorithm 1 are constructed by using `concurrent.futures` module in Python. The inference of the ADTD model is based on PyTorch library (version 1.7.1) with CUDA support. For the Transformers, there are widely-acknowledged implemetations, and in our model we have reused the code from Hugging Face.[2]

6 EVALUATION

This section conducts comparative studies on the Taste framework and previous state-of-the-art approaches: TURL [17] and Doduo [30]. First, we introduce datasets and experimental settings. Then, we define three key performance metrics that can reflect whether a semantic type detection algorithm is suitable for

[2] https://github.com/huggingface/transformers

the cloud environment. Finally, comparison details with respect to these metrics are described one by one.

6.1 Datasets and Settings

6.1.1 Datasets. The datasets we used are WikiTable [17] and GitTables [20], which are well recognized in the field of tabular data understanding, since they encompass rich semantic types from diverse domains. Detailed descriptions of the datasets as well as their key properties are summarized in Table 2.

WikiTable is a corpus of relational tables extracted from Wikipedia, which contains a total of 406,706 tables and 255 semantic types. Each column in the WikiTable is annotated by at least one semantic type. To make the comparison fair, we used the same training/validation/testing splits as TURL (publicly available at [17]).

GitTables is a large-scale corpus of relational tables extracted from CSV files in GitHub, which contains more than 10M tables and nearly 2,000 semantic types covering diverse real-life scenarios. Considering the large data volume in GitTables, which cannot be accommodated by most hardware, we randomly select 100K tables from it (with random seed 0). The selected tables compose a dataset named **GitTables-100K**, and we split it into training/validation/testing sets with a ratio of 80/10/10. In fact, our experiments are conducted on GitTables-100K, but for brevity, we refer it as GitTables in the rest of the paper. Different from WikiTable, GitTables contains columns without any semantic type. For example, 32.13% of columns in the GitTables-100K testing set do not have any semantic types. In this case, we assign a background type (type: null) to these columns.

6.1.2 Method of reading table data. During model training and prediction, our method of reading table content is that we only retrieve m rows from each table, and for each column we find the first n non-empty cell values as model input, where $n \leq m$. We avoid using empty data, as they contribute nothing to semantic type inference. Another pre-processing step is that the input to the DL models that exeeds the sequence length limit should be truncated. Furthermore, we consider two types of table scanning methods: *first m rows* and *random sampling of m rows*. The former is the default configuration for the subsequent experiments, while the latter is used to mitigate the potential impact of uneven data distribution. Besides, the default settings of m and n are 50 and 10, respectively. Later in Section 6.8, we will vary the value of n to study how the input data size affects Taste's performance.

In addition, considering the operation of computing inter-column attention is GPU memory intensive, we will split the tables with large column size into smaller ones during training and prediction, so that the Transformer-based models can be supported by GPU devices of different capacities. Specifically, we introduce a *column splitting threshold* (i.e., l), whose default value is 20 (to fit our GPU device). In other words, wide tables in the datasets will be split into smaller ones with no more than l columns. In Section 6.8, we will evaluate the effect of varying l on the performance.

6.1.3 Training and prediction. The training of Taste's DL models is conducted *on premise*, i.e., on a Linux server with Intel 13th Gen i7-13700K CPU, NVIDIA GeForce RTX 4080 GPU and 64 GB RAM. During the training phase, we first initialize the network weights of their embedding layer and encoder using the same pre-trained checkpoint as TURL [17], which is pre-trained on an unlabeled Wikipedia table corpus [10]. And then for each dataset, we fine-tune all the network weights across the entire model using the corresponding training set for 20 epochs. Under the default settings of n and l, the training time on WikiTable and GitTables is 97 and 66 mins, respectively, and the peak GPU memory usage is 6,954 and 11,136 MB, respectively.

Table 2: Summary of the open datasets

Dataset	# tables	# cols	# types	% col w/o types
WikiTable	406,706	654,670	255	0%
- training	397,098	628,254	255	0%
- validation	4,764	13,025	248	0%
- testing	4,844	13,391	248	0%
GitTables-100K	100,000	1,212,987	1,953	31.56%
- training	80,000	966,107	1,884	31.43%
- validation	10,000	122,331	1,239	31.98%
- testing	10,000	124,549	1,289	32.13%

However, the prediction process is conducted on *real cloud infrastructure* in the State Cloud [6]. Specifically, the Taste models are deployed on an ECS (pi7.8xlarge.4 type with 32 vCPU, 128 GB RAM and 2 NVIDIA A10 GPUs), and the test data are stored in an RDS for MySQL (version 8.0 of general-purpose family with 8 vCPU, 16 GB RAM and 500 GB SSD). Both the ECS and RDS instances reside in one VPC (Virtual Private Cloud), and the average network delay between them is 5 ms. The reason for choosing MySQL is that it is the most popular database type in our cloud as well as in China cloud market. The separation of semantic type detection service and user databases can mimic the real production environment, and also enables us to evaluate the end-to-end execution time of semantic type detection algorithms in a holistic manner. When constructing table schemas in MySQL, we convert the table contextual information in the datasets, such as page titles and section titles in WikiTable, into table comments. Similarly, the information is also used by TURL to aid semantic type detection.

6.2 Baselines and Metrics

We consider the following baselines for performance comparison.

- **TURL [17]**: It utilizes pre-trained multi-layer Transformers of the same size as Taste ($L = 4$, $A = 12$, $W_{max} = 512$, $I = 1200$, $H = 312$, parameter size 14.5M) to encode textual input. The pre-trained encoder captures not only the lexical, semantic, contextual information of words, but also the association between table content and metadata.
- **Doduo [30]**: It jointly considers column type detection and inter-column relation detection, and leverages multi-task learning to train its model. Doduo adopts a larger pre-trained language model, i.e., BERT base ($L = 12$, $A = 12$, $W_{max} = 512$, $I = 3072$, $H = 768$, parameter size 108M).

Despite of using pre-training and fine-tuning paradigm, the baseline approaches and Taste differ in embedding size, model structure and complexity, which will lead to different performance behavior in semantic type detection tasks. To thoroughly evaluate our approach, we produce *six variants* of Taste as follows. This allows us to do an ablation study and evaluate the utility of different components in the Taste framework.

- **Taste**: The default setting of Taste where latent caching and pipelining execution are enabled during inference,

table scanning is based on first m rows and column histograms metadata are not used.

- **Taste with histogram**: Default Taste but with column histograms being part of metadata features. Column histograms may be missing in reality and are only available when users explicitly tell databases to generate such information by running `ANALYZE TABLE` SQL statements. This variant mimics the case when histograms are readily available.
- **Taste without pipelining**: Default Taste but with pipelining execution disabled. As a result, it degenerates to the sequential mode in which tables and stages are processed one by one without overlapping.
- **Taste without caching**: Default Taste but with the latent cache in the metadata tower of ADTD disabled. Then, the two models for P1 and P2 become independent. In some sense, it is equivalent to disabling multi-task learning in our approach. In this way, we can examine the benefit of employing multi-task learning.
- **Taste with sampling**: Default Taste but with the table scanning method replaced by random sampling of m rows (seed assigned to 0 in MySQL `RAND` function).
- **Taste without P2**: The second phase in Taste is disabled by setting $\alpha = \beta = 0.5$; this mode can be applied in strict data privacy settings where users only allow cloud services to access metadata.

From a cloud provider's perspective, we use metrics of the F1 score to evaluate the prediction performance, the execution time to evaluate the efficiency, and the ratio of scanned columns to evaluate the intrusiveness to user databases. This is different from most existing work that mainly focuses on the F1 score. Details of these metrics are explained as follows.

- **Execution time**: It is the overall time cost that a semantic type detection approach pays to finish processing all the columns in the test dataset. We consider the end-to-end time which involves the time for creating/closing database connections, fetching metadata and content from databases, and the prediction time.
- **Ratio of scanned columns**: It is defined as the total number of columns whose contents have been retrieved by the semantic type detection approaches divided by the total number of columns in the test dataset. This metric not only measures every approach's intrusiveness to user databases but also reflects the effectiveness of our metadata-based DL model in finding correct semantic types.

Note that α and β are two important parameters in Taste framework. In the following experiments, we empirically set $\alpha = 0.1$ and $\beta = 0.9$ for all the above variants except Taste without P2. Detailed analysis of α and β settings will be discussed in Section 6.7.

6.3 Execution Time

First, we evaluate the end-to-end execution time for different DL-based semantic type detection approaches on the open datasets. Fig. 4 depicts the average execution times and their standard deviations over ten runs. Since the total number of columns in GitTables dataset is much larger than WikiTable, unsurprisingly the execution times on GitTables are longer.

Compared to TURL and Doduo, Taste reduces the execution time by 40.5% and 52.9%, respectively, on WikiTable, and by 75.4% and 85.0%, respectively, on GitTables. The main reason

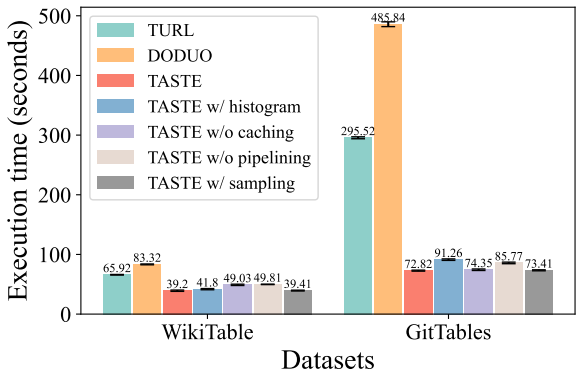


Figure 4: End-to-end execution time of different DL-based semantic type detection approaches.

for the improvement is that semantic types of a large portion of columns can be determined during the first phase of Taste, which is a much less costly process than TURL and Doduo which rely on column content for prediction. Besides, the performance improvement is much higher on GitTables because we observe a higher relevance of the metadata to semantic types on this dataset. As we can see later in Fig. 5, only 45.0% of columns must be scanned in the second phase on the WikiTable dataset, and this number is as low as to 1.7% on GitTables. These results prove not only the efficiency of the metadata-based DL models over the content-based counterparts, but also its effectiveness in saving the costly content retrievial operations.

Fig. 4 also shows that the execution times for Taste with histogram are 6.6% and 25.3% higher than vanilla Taste on WikiTable and GitTables, respectively. Though including histograms in Taste slightly increases the execution time, it still significantly outperforms the TURL and Doduo baselines. Considering histograms are beneficial to improving the F1 score (see Table 3), we suggest that histograms, if available in user data sources, should be used as input for semantic type detection in practice.

In addition, we have found that latent caching can improve the execution efficiency of Taste. Compared to Taste without caching, Taste can reduce the execution time by 20.0% on WikiTable and 2.0% on GitTables. Recall that multi-task learning and parameter sharing between the two towers enables Taste to avoid duplicate computation of the metadata tower. During inference, a higher probability of entering P2 brings a higher benefit from the latent caching mechanism. As in the case of WikiTable, when many uncertain columns need to go through P2, the latent caching mechanism can yield significant execution time reduction.

By comparing Taste and Taste without pipelining, we further demonstrate that pipelining is another effective mechanism to reduce execution time. In the experiment, we set the size of the thread pools ($TP1$ and $TP2$ in Algorithm 1) to 2. Enabling pipelining brings a reduction of execution time by 21.3% on WikiTable and 15.1% on GitTables. We claim that with more powerful hardware and a larger thread pool, we can further expand such performance gain. However, replacing the first m rows scanning method by random sampling only incurs lightweight impact on the execution time, e.g, increase from 39.20s to 39.41s on WikiTable dataset. Such increase is expected because random sampling runs more slowly than sequential scan of the first m rows in MySQL.

Table 3: F1 scores of different DL-based semantic type detection approaches on WikiTable and GitTables datasets ($n = 10$ and $l = 20$). For Taste and its variants, $\alpha = 0.1$ and $\beta = 0.9$.

Model	Precision	Recall	F1
WikiTable dataset			
TURL	0.9275	0.9263	0.9269
Doduo	0.9325	0.9234	0.9279
Taste	0.9344	0.9267	0.9306
Taste w/ histogram	0.9414	0.9267	**0.9340**
Taste w/ sampling	0.9342	0.9271	0.9306
GitTables dataset			
TURL	0.9852	0.9767	0.9809
Doduo	0.9923	0.9873	0.9898
Taste	0.9947	0.9842	0.9894
Taste w/ histogram	0.9957	0.9862	**0.9909**
Taste w/ sampling	0.9945	0.9841	0.9893

6.4 F1 Score

Though execution time and intrusiveness are the primary concern of this work, prediction accuracy is also essential. Table 3 shows F1 scores of various approaches on the datasets. Note that entity-linking information in WikiTable is not used in our experiment for fair comparison. Also, as suggested by Doduo's authors [30], metadata have been incorporated into column values to fit the Doduo framework in our experiments. Note that pipelining and latent caching only affect Taste's runtime efficiency, and thus Taste without pipelining and Taste without caching are not discussed in the context of F1 score comparison.

Although the baselines have achieved very high F1 scores on both datasets, Taste and its variants can still outperform them and achieve state-of-the-art performance. From Table 3, under the setting of $\alpha = 0.1$ and $\beta = 0.9$, Taste achieves F1 scores of 0.9306 and 0.9894 on WikiTable and GitTables, respectively. Furthermore, we observe that column histograms can positively influence the inference quality of our model. When histograms are included in the model training and prediction, Taste with histogram increases the F1 score to 0.9340 and 0.9909, respectively, which are slightly higher than those of TURL and Doduo. However, sampling method almost causes no change to F1 score, indicating our approach's robustness to column content variation.

The reasons for Taste's superior performance in F1 score are mainly two-fold. First, we use more adundant metadata than TURL and Doduo. Apart from textual metadata like table name or comment, we also leverage other database native metadata, such as data type, nullability, various statistics (e.g, max, min, ndv, and histogram), etc. Second, the way of computing attention in Taste is different from that in TURL and Doduo. Specifically, for each cell value, we compute its cross-attention with all column/table metadata (e.g., column/table name) and the cell values in the same column. In contrast, TURL computes the corresponding cross-attention by only considering the current column's metadata. Doduo actually mixes column metadata with column content together as model input, and thus cannot effectively extract the relations between metadata and column content as we do. In short, when computing attention, Taste can better utilize metadata, which is beneficial to the prediction task.

In addition, we have studied the impact of missing column content in the strict data privacy settings. To create this scenario, we replace the column content input with an empty string during the inference for TURL and Doduo, while disabling P2 by setting $\alpha = \beta = 0.5$ for our approach. Evaluation results are summarized in Table 4. Unfortunately, F1 scores of TURL and Doduo on the WikiTable dataset dropped drastically to 0.6153 and 0.5832, respectively, compared with the previous results in Table 3. However, our approach retains a high F1 score, with only a slight decrease from 0.9306 to 0.9047. The main reason for this is that the multi-task learning and asymmetric network structure of ADTD allow us to train the DL model used in P1 purely based on metadata. So removing P2 in the Taste framework does not significantly affect the inference quality of P1. This demonstrates the robustness of Taste under different data privacy settings in the cloud.

Table 4: F1 scores of Taste and the baseline approaches on WikiTable and GitTables datasets when only metadata are used as input ($l = 20$).

Model	Precision	Recall	F1
WikiTable dataset			
TURL w/o content	0.6787	0.5627	0.6153
Doduo w/o content	0.5266	0.6534	0.5832
Taste w/o P2	0.9037	0.9057	**0.9047**
GitTables dataset			
TURL w/o content	0.9855	0.9753	0.9804
Doduo w/o content	0.9893	0.9832	0.9862
Taste w/o P2	0.9941	0.9843	**0.9892**

6.5 Ratio of Scanned Columns

To study the intrusiveness of semantic type detection approaches on user databases, we collect the ratio of columns that each approach needs to scan for column content. Fig. 5 summarizes the results for different approaches. Obviously, this metric is 100% for TURL and Doduo regardless of the dataset, because both of them rely on column content to function. On the WikiTable dataset, we can see from Fig. 5 that the ratio of scanned columns is 45.0% and 43.6% for Taste and Taste with histogram, respectively. On the GitTables dataset, the metric becomes 1.7% and 0.9% for Taste and Taste with histogram, respectively. Since pipelining, latent caching and sampling do not affect the number of columns to scan, the corresponding variants are not shown in the figure. In other words, they have the same ratio of scanned columns as Taste. Additionally, same as our approach, both TURL and Doduo also need to fetch metadata from user databases. If we quantify the overall impact on user databases by how much metadata or column content needs to be scanned, we believe that our approach can significantly mitigate such impact, and thus is more acceptable to cloud users.

6.6 Columns Without Any Types

So far, all the experiments on WikiTable have been conducted with all columns associated with semantic type labels. However, in reality, such situation rarely happens. Instead, the majority of columns are not related to any semantic types, and users are only concerned about a small set of semantic types, such as Personally Identifiable Information (PII) for privacy protection. To this end, we mimic the real-world scenario by removing a certain number of semantic type labels from the WikiTable dataset.

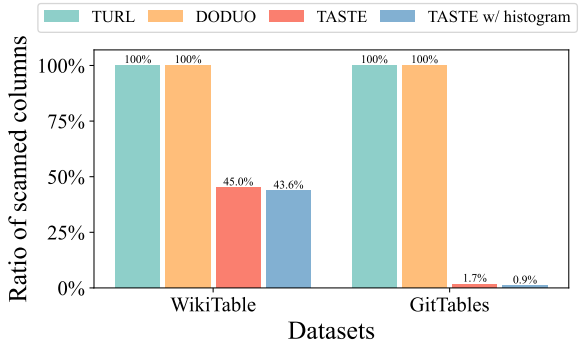


Figure 5: Ratio of scanned columns by different DL-based semantic type detection approaches.

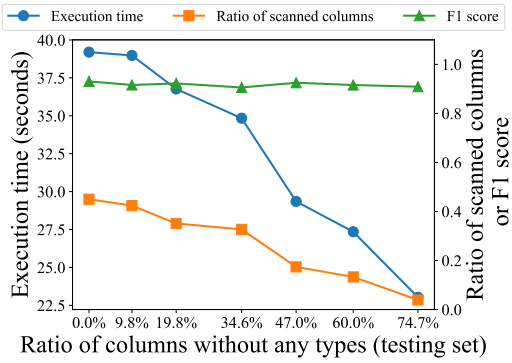


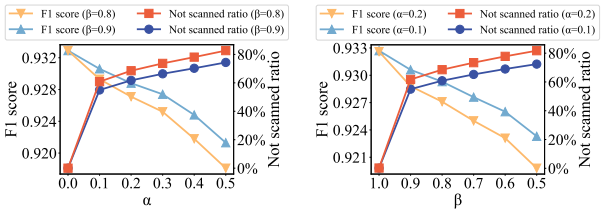
Figure 6: Performance of TASTE when the ratio of columns without any types changes (WikiTable).

First, we randomly select k semantic types (with random seed 0) from the semantic type domain set (S) of WikiTable, which consists of 255 types. These k types compose a new semantic type domain set (S_k), called retained type set. We apply the same split method to obtain training/validation/testing sets from the original WikiTable dataset. But for each column in these sets, we only retain the annotated types that exist in S_k; other types are removed. If a column has no types remaining after the process, we assign it with a background type (type: null). As a result, we can obtain a *tuned dataset*, named WikiTable-S_k. By adjusting the value of k from 50 to 240, we generate a sequence of new datasets, each of which has a different ratio of columns without any types, denoted as η.

Then, we fine-tune our DL models on the tuned datasets and evaluate their performance. Experimental results are depicted in Fig. 6. As we can see from the figure, both execution time and the ratio of scanned columns will drop when η increases. At the same time, the F1 score maintains stably. This trend is reasonable, because columns without any semantic types can be recognized by the DL model in P1, and there is no need to fetch their content in P2. It reflects TASTE's applicability in the real world when only a small portion of columns should be annotated with semantic types.

6.7 Settings of α and β

Now we conduct sensitivity analysis on α and β in the TASTE framework. Fig. 7 shows how the performance metrics (i.e., the F1 score, and ratio of columns that are not scanned) change when



(a) Varying α with fixed β (b) Varying β with fixed α

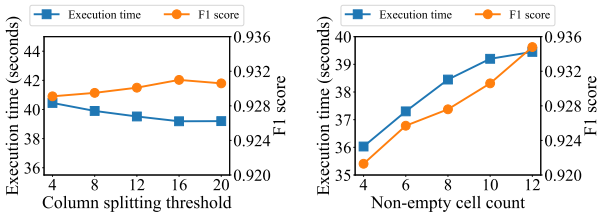
Figure 7: Effects of varying α and β (WikiTable).

the values of α and β vary on the WikiTable dataset. Generally speaking, as α gets smaller and β gets larger, the F1 score increases and the ratio of columns not scanned decreases. The main reason is that a wider (α, β) interval causes more columns in P1 to become uncertain and thus requires an additional P2 phase to verify them. Although involving P2 improves the F1 score, the cost is that the execution time and ratio of scanned columns will increase too. Note that the execution time positively correlates to the scanned ratio, according to Fig. 6. Therefore, we omit to discuss the impact of α and β on the execution time for brevity.

To conclude, though the settings of α and β are application-dependent, we list some rules of thumb as follows. In order to achieve a high F1 score, users need to set small α and large β. If users are sensitive to data exposure, they should shrink the interval of (α, β), or in the extreme case, set $\alpha = \beta$ to disable scanning column content. To well balance the F1 score and execution time/data source intrusiveness, reasonable α and β can be chosen near the cross points of the F1 score curve and the ratio of columns not scanned curve in Fig. 7. In fact, this also reveals another advantage of TASTE, namely, *flexible* to different application scenarios.

6.8 Settings of l and n

As mentioned in Section 6.1, we split big tables with more than l columns into smaller ones, and read only n non-empty cell values to meet the model's input sequence length constraint. So far, we have fixed l and n to 20 and 10, respectively, in the previous experiments. In the following, we will investigate how changing these two parameters affects TASTE's performance. Fig. 8(a) shows the evaluation results when we fix $n = 10$ but vary l from 4 to 20. We can observe an opposite trend for execution time and F1 score, as l becomes larger. Specifically, the execution time continuously drops. This is because smaller l results in more table splitting. The increased table number causes the prediction time to increase, and consequently raises the overall execution time. However, the F1 score will increase, since more columns in tables enable us to utilize more metadata to compute attention, and then improve prediction accuracy. In addition, we have studied how TASTE's performance reacts to the change of n. As shown in Fig. 8(b), both execution time and F1 score increase, as n gets larger (fix l to 20). It implies that the number of input cell values has positive impact on the prediction quality. However, the cost is that the framework needs to process more input data, which slows the prediction process. To choose a reasonable n, we should strike a balance between F1 score and execution time. In our production system, n is set to 10.



(a) Varying l with $n = 10$ (b) Varying n with $l = 20$

Figure 8: The impact of l and n on performance (WikiTable).

7 RELATED WORK

Semantic type detection techniques, which aim to automatically discover semantic types from data, are crucial to modern data management and preparation systems. Over the past decades, there have been a lot of endeavors from industry and academia to solve semantic type detection problems through various algorithms.

One category of prior work focuses on regular expression methods. The search for optimal regular expressions matching data of a semantic type is an NP-hard problem. [18, 27] work on efficient methods to construct/learn regular expressions to discover patterns in a table column. XSYSTEM [21] applies the divide-and-conquer principle to improve efficiency, which divides table content separated by known delimiters into structured tokens and extracts patterns from these tokens in parallel. Besides finding the patterns from divided tokens, Auto-validate [29] prunes a fraction of data to tolerate non-conforming values to increase the prediction recall performance. However, regular-expression-based approaches typically require reading and examining table content, leading to a high computation cost and impact on user databases. Moreover, these approaches intrinsically rely on alphabet statistics of table content and fail to leverage rich tabular context, limiting their prediction performance on semantic types.

Another category of work tries to relate table columns to semantic types via value-overlapping approaches [16, 32, 33] or synthesis approaches [34]. The former approaches select semantic types for a target table column based on how its content overlaps with other columns in a knowledge base. Similar to regular-expression-based approaches, they typically requires a full scan of the table column to improve accuracy. Relying on identical matching, they also fail to consider contextual and semantics information of text content. The latter approaches search for or synthesize validation functions to verify if a table column is matched to any semantic types. These synthesized functions usually incorporate domain knowledge and are only applicable to specific semantic types whose content obeys certain protocols or standards, e.g., ISBN and credit card numbers. On the contrary, the Taste framework endeavors to develop generic approaches to support a wide range of semantic types.

In recent years, deep learning approaches for table column type detection have become a promising trend. Sherlock [19] trains a deep neural network with abundant features, including global and character-level statistical features, word embeddings, and paragraph embeddings. Sato [35] includes table context and relationships among neighboring columns in the deep learning model. Some studies [36, 37] use pre-trained language models to leverage the contextual semantics learned from large corpora to significantly improve the prediction performance with less training effort. TURL [17] fine-tunes a pre-trained Transformer-based network and learns column semantic types from table contextual information. Doduo [30] extends the work by adopting multi-task learning into the model to share the knowledge learned from column semantic types and relation tasks. Taste also leverages the pre-trained language model and multi-task learning but with different objectives. These techniques fit into a two-phase framework in order to solve practical issues like runtime efficiency and user acceptance.

Lastly, there are also some hybrid deep learning models that incorporate domain knowledge. Chen et al. [14] propose a hybrid structure with an attentive BiRNN for word embedding, a look-up model for knowledge-based properties extraction, and a convolutional network for feature learning and classification. ColNet [13] ensembles the results from a convolutional neural network and a knowledge-based look-up model to determine the final column semantic types. Our approach is orthogonal to this category of work because we focus on optimizing DL-based approaches for cloud context, and the models in Taste framework are open to integrating the domain knowledge.

8 CONCLUSION AND FUTURE WORK

In this paper, we propose a practical DL-based semantic type detection framework, Taste, for cloud service providers, who raise critical concerns about runtime efficiency and impact on user data sources. Taste consists of a mandatory metadata-driven detection phase and an optional full data-driven detection phase. Taste is efficient and only produces a lightweight impact on user databases for two reasons. First, it leverages various metadata to determine semantic types and avoid costly table scan operations whenever possible. Second, we utilize latent caching to avoid duplicate model computations and pipelining to maximize parallelism among different tables. We develop a novel Asymmetric Double-Tower Detection model based on multi-task learning for the detection in both phases. Taste outperforms prior state-of-the-art approaches in various aspects, including the execution time, F1 score, and intrusion into user data sources. Taste is robust under different data privacy settings and has the potential for cloud-scale deployment.

In the future, we plan to work in the following promising directions. First, we will explore efficient methods to extend the solution to accommodate new semantic types when there are regular updates to the domain set. Second, we plan to make the model adaptable to user feedback about detection results. Third, we will explore seamlessly integrating domain-specific and user-defined semantic types into our framework.

REFERENCES

[1] 2024. *Alibaba Cloud Data Security Center.* https://www.alibabacloud.com/product/sddp/
[2] 2024. *AWS Glue.* https://aws.amazon.com/glue/
[3] 2024. *Google Looker Studio.* https://developers.google.com/looker-studio/connector/semantics/
[4] 2024. *Microsoft Power BI.* https://www.microsoft.com/en-us/power-platform/products/power-bi/
[5] 2024. *Microsoft Purview.* https://azure.microsoft.com/products/purview/
[6] 2024. *State Cloud, China Telecom.* https://www.ctyun.cn/
[7] 2024. *Tableau.* https://www.tableau.com/en-gb/
[8] 2024. *Trifacta Type System.* https://help.alteryx.com/AAC/en/trifacta-classic/concepts/feature-overviews/overview-of-the-type-system.html/
[9] Panagiotis Antonopoulos et al. 2020. Azure SQL Database Always Encrypted. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data.* Portland, OR, USA, 1511–1525.
[10] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. 2015. TabEL: Entity linking in web tables. In *International Semantic Web Conference.*

Springer, 425–441.
[11] Tom B. Brown et al. 2020. Language Models are Few-Shot Learners. *CoRR* abs/2005.14165 (2020). arXiv:2005.14165
[12] Michael Cafarella, Alon Halevy, Hongrae Lee, Daisy Zhe Wang, and Eugene Wu. 2018. Ten Years of WebTables. In *Proceedings of the VLDB Endowment*, Vol. 11. 2140–2149.
[13] Jiaoyan Chen, Ernesto Jiménez-Ruiz, Ian Horrocks, and Charles Sutton. 2019. Colnet: Embedding the semantics of web tables for column type prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 29–36.
[14] Jiaoyan Chen, Ernesto Jiménez-Ruiz, Ian Horrocks, and Charles Sutton. 2019. Learning semantic annotations for tabular data. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. 2088–2094.
[15] Aakanksha Chowdhery et al. 2023. PaLM: Scaling Language Modeling with Pathways. *Journal of Machine Learning Research* 24, 240 (2023), 1–113.
[16] Eli Cortez, Philip A Bernstein, Yeye He, and Lev Novik. 2015. Annotating database schemas to help enterprise search. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1936–1939.
[17] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. TURL: table understanding through representation learning. In *Proceedings of the VLDB Endowment*, Vol. 14. VLDB Endowment, 307–319.
[18] Yeye He, Jie Song, Yue Wang, Surajit Chaudhuri, Vishal Anil, Blake Lassiter, Yaron Goland, and Gaurav Malhotra. 2021. Auto-Tag: Tagging-Data-By-Example in Data Lakes. arXiv:cs.DB/2112.06049
[19] Madelon Hulsebos et al. 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*. 1500–1508.
[20] Madelon Hulsebos, Çağatay Demiralp, and Paul Groth. 2023. GitTables: A Large-Scale Corpus of Relational Tables. In *Proceedings of the ACM on Management of Data*, Vol. 1. 1–17.
[21] Andrew Ilyas, Joana MF da Trindade, Raul Castro Fernandez, and Samuel Madden. 2018. Extracting syntactical patterns from databases. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 41–52.
[22] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for Natural Language Understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. 4163–4174.
[23] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive Visual Specification of Data Transformation Scripts. In *Proceedings of the sigchi conference on human factors in computing systems (CHI)*. ACM, 3363–3372.
[24] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT*. 4171–4186.
[25] Lukas Liebel and Marco Körner. 2018. Auxiliary Tasks in Multi-task Learning. *CoRR* abs/1805.06334 (2018). arXiv:1805.06334 http://arxiv.org/abs/1805.06334
[26] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. 2010. Annotating and searching web tables using entities, types and relationships. In *Proceedings of the VLDB Endowment*, Vol. 3. VLDB Endowment, 1338–1347.
[27] Saswat Padhi, Prateek Jain, Daniel Perelman, Oleksandr Polozov, Sumit Gulwani, and Todd Millstein. 2018. FlashProfile: a framework for synthesizing data profiles. *Proceedings of the ACM on Programming Languages* 2, OOPSLA (2018), 1–28.
[28] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.
[29] Jie Song and Yeye He. 2021. Auto-Validate: Unsupervised Data Validation Using Data-Domain Patterns Inferred from Data Lakes. In *Proceedings of 2021 International Conference on Management of Data (SIGMOD)*. ACM, 1678–1691.
[30] 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*. 1493–1503.
[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).
[32] 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. *Proceedings of the VLDB Endowment* 4, 9 (2011).
[33] Jingjing Wang, Haixun Wang, Zhongyuan Wang, and Kenny Q Zhu. 2012. Understanding tables on the web. In *Conceptual Modeling: 31st International Conference ER 2012, Florence, Italy, October 15-18, 2012. Proceedings 31*. Springer, 141–155.
[34] Cong Yan and Yeye He. 2018. Synthesizing Type-Detection Logic for Rich Semantic Data Types using Open-source Code. In *Proceedings of 2018 International Conference on Management of Data (SIGMOD)*. ACM, 35–50.
[35] Dan Zhang, Madelon Hulsebos, Yoshihiko Suhara, Çağatay Demiralp, Jinfeng Li, and Wang-Chiew Tan. 2020. Sato: contextual semantic type detection in tables. In *Proceedings of the VLDB Endowment*, Vol. 13. VLDB Endowment, 1835–1848.
[36] Ziqi Zhang. 2017. Effective and efficient semantic table interpretation using tableminer+. *Semantic Web* 8, 6 (2017), 921–957.
[37] Chen Zhao and Yeye He. 2019. Auto-em: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning. In *The World Wide Web Conference*. 2413–2424.