T2: Two-phase Tagging of Relational Data in the Cloud Environment

Tao Li*✉
Database Group
eSurfing Cloud, China Telecom
lit51@chinatelecom.cn

Feng Liang*†
Artificial Intelligence Research Institute
Shenzhen MSU-BIT University
fliang@smbu.edu.cn

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

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

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

Runhuai Huang
Database Group
eSurfing Cloud, China Telecom
huangrh@chinatelecom.cn

Xiping Hu†✉
Artificial Intelligence Research Institute
Shenzhen MSU-BIT University
huxp@smbu.edu.cn

ABSTRACT

Over the last decade, we have witnessed the popularity of auto-tagging techniques to help enterprises automatically discover semantic tags from big data. Recently, such auto-tagging capability has also been integrated by major cloud providers into their cloud services. However, when adopted by cloud services, existing auto-tagging approaches can suffer from many practical issues regarding efficiency, extensibility, and impact on user data sources because those services need to support a large and evolving tag set and a huge number of tables/columns from diverse customers. To tackle these issues, we propose a novel auto-tagging framework called T2 for relational data, which are major enterprise data forms in the cloud. T2 consists of two phases, *filter phase* and *verification phase*, to explore the tag space efficiently and find accurate tag results. First, the filter phase of T2 fine-tunes an effective filtering model based on pre-trained deep language models to find a small number of candidate tags from the entire tag set. As a result, irrelevant tags can be filtered out from further verification, thus greatly improving tagging efficiency. Second, our filtering model in the first phase is purely based on database native metadata, thus only generating lightweight metadata retrieval workloads on user databases. Third, the verification phase can leverage various tag-specific models to accurately verify every candidate tag, thus removing possible false results. Furthermore, T2 can be extended online to support new tags by simply skipping the first phase. We develop an efficient batch mode implementation of our framework [1] and evaluate it extensively on open datasets. Experimental results show that T2 saves up to 73.5% intrusive table scan operations and 25.2% execution time while achieving a higher $F1$ score compared with the state-of-the-art approach TURL.

[1]Code available via https://github.com/ncols-bytes/T2-framework.git

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

1 INTRODUCTION

Undoubtedly, data have become one of the most valuable assets for enterprises. To better understand, organize, protect, and analyze data, annotating them with semantic labels is very important. These annotations are known as semantic types, catalogs, or domains in the literature [14, 20, 27], and we refer to them as *tags* [13] in this paper. Proper tags can drive various downstream data applications, such as data asset discovery and searching, sensitive data recognition and protection, data analysis, and machine learning pipelines. As such, almost all mainstream cloud service providers have integrated table column tagging capabilities in their products to help users manage databases and data lakes, e.g., Microsoft Purview [4], AWS Glue [2], Google Cloud Dataplex [3], and Alibaba Cloud Data Security Center [1].

Relying on human effort to tag all table columns is obviously impractical in the big data era and cloud environment, as cloud providers need to serve a large number of customers, each potentially owning a lot of tables. Instead, we utilize various algorithms, from simple regular expressions to complex machine learning algorithms, to achieve automatic tagging (i.e., *auto-tagging*). However, developing cloud-scale auto-tagging algorithms is still challenging, as we need to consider four requirements. 1) **Accurate**: Incorrect tags introduce data quality issues and can adversely impact the downstream data applications. For example, machine learning pipelines can be contaminated if incorrect tags are utilized as input features. 2) **Low intrusiveness**: Content-based tagging algorithms, which require scanning users databases and processing table content, are more intrusive than pure metadata-based algorithms, which only fetch database metadata. Scanning data can increase database I/O and potentially disrupt user business. 3) **Efficient**: We confront the large-scale issue in the cloud environment; that is, there are a vast number of columns to tag (e.g., ten billion-level) from diverse customers. From cloud providers' perspective, it is crucial to have tagging algorithms run efficiently to save computation cost and execution time. 4)

Extensible: New tags are added from time to time as customers propose new requirements, and the tagging algorithm should be easily extended to support them. Such tags are called *emerging tags* in the paper. It is preferable to the cloud service that its tagging capability can be upgraded online to support emerging tags.

Unfortunately, existing approaches to auto-tagging cannot meet some of the above requirements in the cloud environment. They can be categorized mainly into *knowledge-based* and *machine learning-based (ML-based)*. The first category includes regular expression [22], keyword matching, dictionary look-up, and validation function methods. These methods are essentially domain-knowledge-driven; they construct certain recognition models for each tag based on a priori knowledge. For example, a keyword set {address, addr} can be constructed to recognize an address column if the column name contains one of the keywords; ISBN (the international standard book number) and credit card numbers follow certain encoding standards and thus can be validated by predefined functions [27]. However, such models are bound to specific tags and only cover a limited set. In other words, these methods are not generic and would fail to meet requirement 1 when applied to other tags that cannot be easily distinguished from domain knowledge. Furthermore, for each column, it is inevitable for the cloud service to exhaustively evaluate all the models to determine if a column matches any of these tags. Given the large number of columns and tag models, they become very costly and impractical, and thus fail to meet requirement 3. The ML-based approach, on the other hand, can overcome the above limitations. It trains a single model for a batch of tags and has shown great potential in improving tagging accuracy. Sherlock [14] extracts 1,588 features from each column and uses a deep neural network to identify the tags. TURL [12] employs pre-trained Transformer-based language models to further improve the tagging accuracy over Sherlock. However, the ML-based approach usually fails to meet requirements 2 and 4. On one hand, they rely on extracting excessive features from column metadata and content and thus have to scan user databases. On the other hand, extending these machine learning models to support new tags requires retraining a new model, which cannot be done promptly online.

To tackle the challenges of tagging relational data in the cloud scale, we propose *Two-phase Tagging* (T2) to achieve the accuracy, efficiency, low intrusiveness, and extensibility properties in one framework. T2 consists of two phases: *filter* and *verification*, where the filter phase uses a metadata-based machine learning model to find candidate tags for a table column efficiently, and the verification phase determines whether a candidate tag should be finally accepted by further examining column content. The rationale under this two-phase approach is two-fold. First, we find that the majority of user database columns (e.g., 70%) are unrelated to any tags. Second, database native metadata (e.g., column name, comments, data type, and histogram) contain rich information that can be utilized to filter out these unrelated tags. So, filtering can be very effective in helping many columns without tags avoid the second phase of tag verification, which requires expensive table scan operations to retrieve column content. As reading metadata from databases is lightweight and almost all databases provide standard APIs or SQL, the filter phase only introduces limited impact on user databases. Another advantage is that we need to examine only a small subset of tags instead of the entire tag set in the verification phase, thus improving the tagging efficiency substantially.

To achieve high filtering performance, we construct a deep learning model with fine-tune pre-trained Transformer-based language models. The verification phase then is enabled only if the candidate tag set is not empty. In this phase, existing tagging algorithms (knowledge-based or ML-based) can serve as verifiers to determine whether the candidate tags should accepted as final result. Furthermore, emerging tags can easily fit into T2 framework by ignoring the first phase, and get verified directly. By the next release of our cloud service, the filtering model will be retrained and upgraded with emerging tags supported.

We have evaluated T2 extensively on open datasets and evaluation results have shown that T2 achieves the new state-of-the-art $F1$ score and saves up to 73.5% of table scan operations and 25.2% execution time, compared with the previous state-of-the-art approach TURL [12]. Our main contributions of the paper are summarized as follows.

- We propose T2, a framework to solve various practical problems stemming from the large scale tags and databases in the cloud.
- T2 fine-tunes a novel filtering model based on pre-trained deep language models to effectively reduce the tag set to verify, thus significantly improving overall tagging efficiency. We develop a batch mode implementation of T2 framework to further improve runtime efficiency.
- T2 embraces the existing knowledge-based and ML-based tagging algorithms as verifiers and can be extended easily to support emerging tags.

The rest of the paper is organized as follows. Section 2 gives some preliminaries about auto-tagging techniques and the auto-tagging problem in the cloud environment. Our approach, called T2 framework, is described and theoretically analyzed in Section 3. We introduce an efficient implementation of T2 framework in Section 4. Evaluations and comparisons with existing work are conducted in Section 5, which is followed by a summary of related work in Section 6. We conclude the paper with directions of future work in Section 7.

2 BACKGROUND AND SCENARIO

This section introduces various features that existing methods commonly use to tag relational data, the classification of tagging algorithms, and the challenging auto-tagging problem in the cloud environment.

2.1 Column Features

Enterprise relational data commonly exist in relational databases, such as OLTP systems (e.g., MySQL, PostgreSQL) and data warehouses (e.g., AWS Redshift, Snowflake, Hive, SparkSQL). Tagging relational data often focuses on column level, and the features that can be utilized for column tagging mainly include metadata and data (called content interchangeably in this paper).

Generally, there are three types of metadata in the realm of relational databases [7]:

- **Basic metadata**: the table name, column name, data type, and index type.
- **Statistics metadata**: the maximum, minimum, number of distinguished values, and statistical histogram.
- **Advanced metadata**: the correlation and functional dependency inferred by data profiling techniques.

Basic metadata are schema information, an integral part of database systems. Statistics metadata are crucial to cost-based

Table 1: Column related metadata in MySQL and SparkSQL

Metadata Type	MySQL	Spark
Column name	✓	✓
Data type	✓	✓
Column comments	✓	✓
Nullable	✓	
Index name	✓	
Index type	✓	
Index comments	✓	
Column sequence in index	✓	
Partition		✓
Index cardinality	✓	
Max		✓
Min		✓
Number of nulls	✓	✓
Number of distinct values		✓
Avg length		✓
Max length		✓
Histogram type	✓	
Histogram sampling rate	✓	
Number of buckets	✓	✓
Height		✓
Bucket values	✓	✓

optimizers (CBO), so modern databases are able to collect those statistics for CBO to use. For example, MySQL [5] and SparkSQL [6] provide the `ANALYZE TABLE` query to collect and update statistics stored inside these systems. In contrast, advanced metadata are not maintained inside databases and need external profile systems to perform complex analysis. Therefore, advanced metadata are generally unavailable, and not discussed in this paper. Table 1 shows the column metadata that MySQL of 8.0.x and SparkSQL of 3.4.x maintain in their engines.

Regarding column content, each cell value can be treated as an input to tagging algorithms, e.g., to evaluate regular expressions or validation functions. ML-based tagging algorithms, however, prefer to extract fine-grained features from column metadata and values. For example, Sherlock [14] extracts 960 features related to character-level distributions, 200 features by pre-trained word embeddings, and 400 features from self-trained paragraph vectors.

Generally speaking, collecting metadata is more lightweight operation and causes less impacts on user databases compared with reading data from databases. Database systems commonly provide APIs or SQL queries to facilitate metadata retrieval, e.g., JDBC APIs, `SELECT * FROM information_schema.columns` in MySQL, and `DESCRIBE EXTENDED tb_name col_name` in SparkSQL. Thus, tagging algorithms purely relying on metadata are less intrusive to user databases.

2.2 Column Tagging Algorithms

Over the last decade, many tagging algorithms have been proposed to help enterprises discover various types of tags for data, ranging from simple keyword matching to complex machine learning models. These algorithms have been widely adopted in commercial BI tools, metadata management software, and cloud data governance services in recent years. From our viewpoint, existing algorithms can be generally categorized into two types: *knowledge-based* and *machine learning-based (ML-based).*

Knowledge-based algorithms include regular expression, keyword matching, dictionary look-up, bloom filter and validation functions. The commonality of these algorithms is that they all rely on a priori domain knowledge, and the difference among them is just the form of expressing domain knowledge. For example, regular expressions can effectively characterize textual data with specific string patterns, such as email address, date time, and machine-generated data [22]; dictionaries can be utilized to recognize columns with finite value sets like country, football league, and Turing Award winner. Such knowledge is widely spread in various domains and industries. The features that knowledge-based algorithms utilize mainly include basic metadata and column content.

In contrast, ML-based algorithms are more complicated and suitable in cases where knowledge cannot be explicitly constructed. A typical example is to recognize person names. In recent years, deep neural networks and pre-trained Transformer-based language models (e.g., BERT [18] and ERNIE [24]) have been applied in the auto-tagging domain for general cases and found to be very promising. The state-of-the-art in this lane are Sherlock [14] and TURL [12]. Those approaches utilize excessive features extracted from basic metadata and column content.

2.3 Column Tagging in the Cloud

Even though the performance of tagging algorithms (e.g., tagging accuracy) has been continuously improved over the last decade, existing approaches have problems when applied in the cloud environment because they do not address many practical concerns such as intrusion into user databases, efficiency, and extensibility mentioned in the Introduction.

Databases are critical infrastructure of enterprises, and it is unacceptable for most customers that the auto-tagging service introduces significant overhead on their databases. ML-based algorithms, like Sherlock and TURL, require scanning user databases to retrieve every column's data and are thus undesirable. Besides, cloud services need to deal with the tagging of a large number of columns from customers from diverse sectors and industries. Execution efficiency then becomes a paramount concern. It is preferable for the algorithm to be able to scale to a large number of tags and columns. Knowledge-based algorithms, tied to specific types of tags due to their knowledge-driven nature, are generally not scalable. In the worst case, our auto-tagging service must exhaustively run all the knowledge-based algorithms to find possible tags for every column, which is very costly in the case of large tag set.

At last, there are ad-hoc requirements from customers to support new tags. Our cloud service needs to deliver the new capability as early as possible to improve user satisfaction. To distinguish the tags already supported by the current release of our service and the new tags, we define two types of tags as follows.

*Definition 2.1 (**Published Tag**).* A tag t is called *published* if the current release of our tagging service supports the tag.

We denote the set of published tags as T. Note that the size of T can be huge and is constantly increasing as the cloud service evolves to cover more and more tagging scenarios.

*Definition 2.2 (**Emerging Tag**).* A tag t is called *emerging* if it appears after the release of our service, i.e., $t \notin T$.

We denote the set of emerging tags as T^e, where $T \cap T^e = \emptyset$, and the overall tags our service needs to deal with is $T \cup T^e$. For easy reference, key notations in the paper are listed in

Table 2: Notations for the multi-column multi-tag training problem and model formalization

Notation	Description
T	tag domain set
C	column set $\{c\}$
c	a column in C characterized by $(\mathcal{M}_c, \mathcal{D}_c)$
$\mathcal{M}_c$	column c's metadata
$\mathcal{D}_c$	column entity cell content
$\hat{T}_c$	candidate tags predicted by the filtering model for c
T^e	emerging tags not supported by the filtering model
T^{1F}	output tags of the filter phase
T^{2F}	output tags of the verification phase
$f(c)$	filtering model with input column c
$V_t(c)$	verification model associated with tag t for input column c

Table 2. Emerging tags are inevitable in the real world mainly because the service product generally follows a regular release cycle, e.g., one release every two weeks. During the gap between consecutive releases, it is preferable that the service be extended online to support emerging tags. Unfortunately, to the best of our knowledge, none of the existing approaches address this concern.

3 OUR APPROACH: THE T2 FRAMEWORK

This section first introduces the framework of T2 and the details of its components. It then formalizes the two-phase tagging problem and analyzes its performance with discussion of the performance impact factors.

3.1 Overview

Figure 1 illustrates an overview of the T2 framework. It consists of two phases: the filter phase and the verification phase. The filter phase eliminates the columns that need not be tagged and the tags unrelated to a particular column. As a result, it reduces the number of tags that need to be verified in the verification phase. The verification phase determines the ultimate tags for a column based on the results of a set of verifiers associated with each tag. The ultimate column tags can then be used for downstream data applications.

The filter phase applies a filtering model to find a reduced set $\hat{T}$ from the published tag set. Inspired by existing work like [12, 14, 23], the filtering model is a deep neural network consisting of multiple pre-trained Transformer-based layers to encode textual features. It outputs the matching probabilities of the published tags. and selects the output tags with high probability in a top-K or threshold-based manner. Emerging tags not yet supported by the filtering model will be considered candidate tags directly for the subsequent verification phase.

*Definition 3.1 (**Candidate Tag**).* A tag $t \in T$ is called a candidate tag for column c if the filter phase selects t as column c's potential tag, which will be verified in the verification phase.

We denote the set of all candidate tags for column c by T_c^{1F}. Each candidate tag belongs to either the published tag set or the emerging tag set, i.e., $T_c^{1F} \subset T \cup T^e$. Note that the filter phase uses only database native metadata including basic and statistics metadata (described in Section 2.1) rather than column data as the model inputs, for efficiency and intrusiveness considerations. Combining metadata and a deep pre-trained language model can achieve both the minimum impact on user databases and the high filtering effect of unrelated tags. Also, we notice that some types of metadata (see Table 1) may not always be available. For example, column histogram information is missing unless the `ANALYZE TABLE` statement has been executed for that column. In this case, features with missing metadata values are set to null values when constructing input to the filter model.

At the verification phase, every candidate tag is carefully examined by its associated verifier. Existing tagging algorithms, either knowledge-based or ML-based, can serve as verifiers. In this way, we can inherit tremendous progress made during the past decades and accommodate these algorithms into our framework. A multi-label verifier can support multiple candidate tags, indicating an n-to-1 mapping from the candidate tags to the associated verifier, where n can be 1. Specifically, we verify candidate tags separately based on the type of tags. For knowledge-based tags, we check them one by one by the associated tagging algorithms. For ML-based tags, we develop another verification model based on deep pre-trained language models similar to the filtering model, except that the verification model leverages features from column content in addition to metadata. The detailed filtering and verification model design will be described in the subsequent subsections. Therefore, table scan operations are only needed in the verification phase. The verification phase is skipped when a column has no candidate tags in the filter phase. In the case of a large table, we only sample the table content to reduce I/O and network impact on user databases and the scanning time.

Candidate tags which pass the verification are called accepted tags. The accepted tags with respect to column c are denoted by T_c^{2F}.

*Definition 3.2 (**Accepted Tags**).* For a candidate tag $t \in T_c^{1F}$, it is called accepted for column c if the tag is verified true in the verification phase.

3.2 Problem Statement

We formalize the *two-phase model training problem* and perform an in-depth discussion about how to build models appropriately as follows.

3.2.1 Training problem. Given a dataset which contains a set of columns C, a tag domain set T, and a set of labels (tags) T_c for each column $c \in C$, the two-phase model training training problem is to find the filtering model and verification model such that the overall $F1$ score of the T2 framework is maximized. The overall $F1$ score is defined in Formula 4 subsequently.

Let us denote the filtering model as f, and the output of the model is

$$\hat{T}_c = f(\mathcal{M}_c),$$

where $\mathcal{M}_c$ is the metadata features of column c. There is also a set of emerging tags not supported by the filtering model, denoted by T^e. The output of the filter phase is the union of $\hat{T}_c$ and T^e, that is,

$$T_c^{1F} = \hat{T}_c \cup T^e.$$

The verification phase then verifies each tag $t \in T_c^{1F}$. Knowledge-based verifiers and ML-based verifiers are evaluated separately due to their different nature. The output of the verification phase (i.e., accepted tag set) can be represented as:

$$T_c^{2F} = \{t | V_t(c) \text{ is true},\ t \in T_c^{1F}\},$$

where $V_t(c)$ is the verifier with respect to tag t.

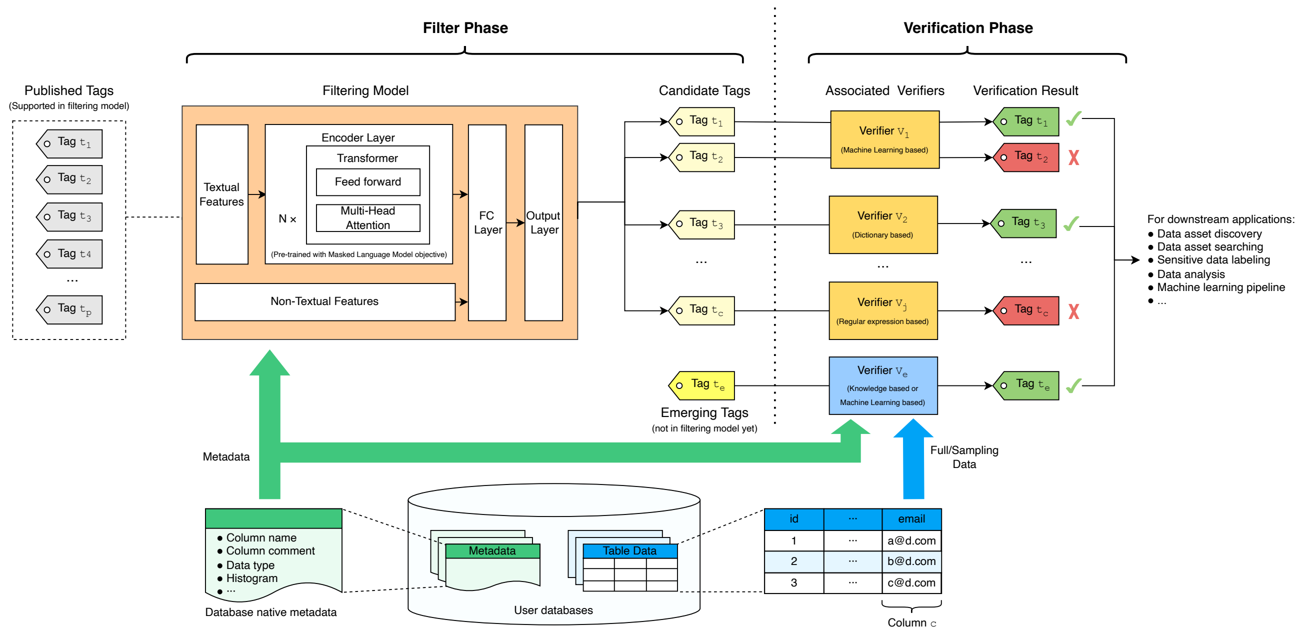

Figure 1: An overview of the T2 framework

Table 3: Notations for performance analysis

Notation	Description
P	# positive tags of a column
TP	expected # true positive predictions by T2
FN	expected # false negative predictions by T2
FP	expected # false positive predictions by T2
T_{TP_0}	true positive tag set of the filtering output
T_{FP_0}	false positive tag set of the filtering output
$recall_t$	recall of verifier V_t
fpr_t	false positive rate of verifier V_t
$F1$	$F1$ score of the T2 prediction

3.2.2 Performance analysis. We formally define the performance metrics of the T2 framework, including recall, precision, and $F1$ score in the following. We also discuss the influential factors to these performance metrics to guide the building and selection of the best models in different phases of the T2 framework. Table 3 summarizes the notations used in this subsection for easy reference.

For a table column input, suppose the number of positive tags is P, and the expected number of true positive, false negative, and false positive tags derived by the framework are TP, FN, and FP, respectively.

In the whole procedure, a true positive tag is an actual positive one predicted positive by both the filter phase and the associated verifier in the verification phase. Therefore, the expected number of true positive tags predicted by the framework is the sum of the expected true positive rate (recall) of all the verifiers of the true positive tags of the filter phase. We have

$$TP = \sum_{t \in T_{TP_0}} recall_t, \tag{1}$$

$$FN = P - TP, \tag{2}$$

where T_{TP_0} is the set of true positive tags of the filter phase and $recall_t$ is the recall of the verification model V_t in the verification phase.

Similarly, in the whole procedure, a false positive tag is an actual negative one predicted positive by both the filter phase and the associated verifier in the verification phase. Therefore, the expected number of true positive tags predicted by the framework is the sum of the expected false positive rate of all the verifiers of the false positive tags of the filter phase. We have

$$FP = \sum_{t \in T_{FP_0}} fpr_t, \tag{3}$$

where T_{FP_0} is the set of false positive tags of the filter phase and fpr_t is the false positive rate of the verification model V_t in the verification phase.

By Formulae 1, 2, and 3, the expected values of the recall, precision, and $F1$ score of the framework can be derived as follows:

$$Recall = \frac{TP}{TP + FN} = \sum_{t \in T_{TP_0}} recall_t / P,$$

$$Precision = \frac{TP}{TP + FP} = \frac{\sum_{t \in T_{TP_0}} recall_t}{\sum_{t \in T_{TP_0}} recall_T + \sum_{t \in T_{FP_0}} fpr_t},$$

$$F1 = \frac{2TP}{P + TP + FP} = \frac{2 \sum_{t \in T_{TP_0}} recall_t}{P + \sum_{t \in T_{TP_0}} recall_t + \sum_{t \in T_{FP_0}} fpr_t}. \tag{4}$$

By looking at Formula 4, we have the following observations. In the filter phase, with other factors unchanged, increasing the number of true positive predictions ($|T_{TP_0}|$) or decreasing the number of false negative predictions ($|T_{FP_0}|$) can increase the overall $F1$ score of the framework. Note that increasing the recall of the filtering model is not a sufficient condition to increase the overall $F1$ score, as a model may increase $|T_{TP_0}|$ and $|T_{FP_0}|$ simultaneously to achieve higher recall. Increasing the precision of the filtering model is not such a sufficient condition either, as a model may decrease $|T_{TP_0}|$ and $|T_{FP_0}|$ simultaneously to

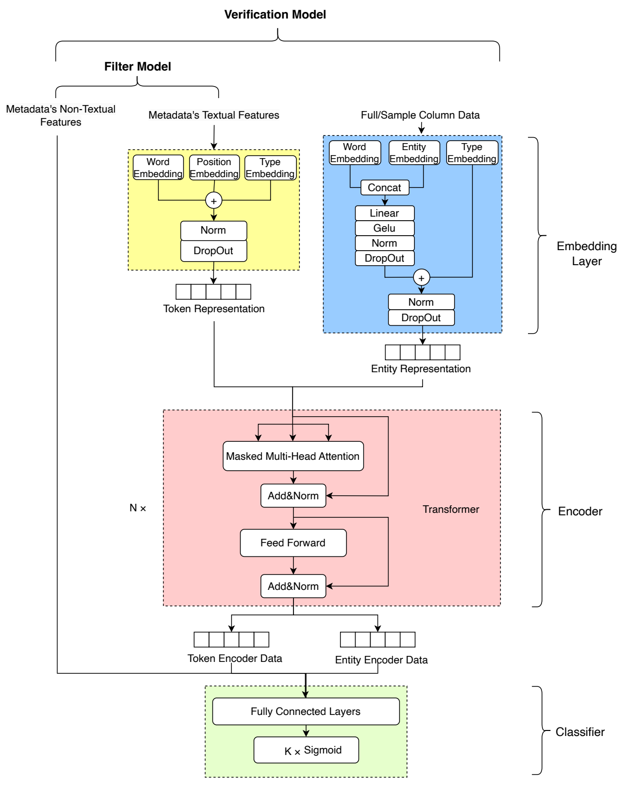

Figure 2: Structure of the filtering and verification models

achieve higher precision. In the verification phase, with other factors unchanged, increasing the recall of the verifiers of the true positive tags of the filter phase ($recall_t, t \in T_{TP_0}$) or decreasing the false positive rate of the verifiers of the false negative tags of the filter phase ($fpr_t, t \in T_{FP_0}$) can lead to the increase of the overall $F1$ score. Or loosely, increasing the recall or decreasing the false positive rate of the verifiers in the verification phase may increase the $F1$ score of the framework.

When emerging tags are added to the framework, they bypass the filtering model and are fed into the verification phase directly. Both $|T_{TP_0}|$ and $|T_{FP_0}|$ may increase, and the expected $F1$ score depends on the recall and false positive rate of the associated verifiers.

To achieve higher framework overall $F1$ score, we have the following claims and suggestions that guide our work. First, A filtering model cannot be chosen merely based on higher precision or recall. Tuning a filtering model for higher overall $F1$ score is a systematic work and is a compromise between higher recall and higher precision. A filtering model could be adjusted with various precision and recall results and integrated into the whole framework for performance evaluation. Second, when multiple verifying models are feasible for a specific tag, the one with higher recall and lower false positive rate should be considered the best for that tag.

3.3 The Filtering Model

In the following, we describe the detailed design of our filtering model. On one hand, it is inspired by the latest work of combining neural network and pre-training, which have proven to be very effective in tagging tabular data. On the other hand, it follows the guide from previous performance analysis. The architecture of the filtering model is composed of an embedding layer, an encoder network, and a classifier network, as shown in Figure 2, which is a multi-label classifier.

The embedding layer transforms the column information to fixed-size embeddings as inputs to the encoder network. Given a column c, $\mathcal{M}_c$ and $\mathcal{D}_c$ represent column metadata and data content, respectively. $\mathcal{M}_c$ can be further divided into $\mathcal{M}_c^t$ and $\mathcal{M}_c^n$, which represent the column's textual metadata and non-textual metadata. The output of the embedding layer is

$$Embed(c) = Embed(\mathcal{M}_c^t). \tag{5}$$

The column textual metadata c_m includes the table name, table description, column name and column comments. The output dimension of the embedding layer is 312.

The encoder network is a pre-trained TinyBERT [17] encoder implemented in TURL [12]. It consists of 4 bidirectional Transformer layers, in each of which the number of self-attention heads is 12, the intermediate size is 1,200, and the hidden size is 312 (L=4, H=312, I=1200, A=12). We do not describe the structure of the well-known Transformer layer in detail. The encoder network is pre-trained on an unlabeled Wikipedia table corpus [8] adopting the Masked Language Model (MLM) objective from BERT to capture the lexical, semantic, and contextual information of texts. Given a sequence of column metadata tokens as the input, the MLM objective masks some tokens at random and tries to recover the masked tokens. The output of the encoding an embedding x is:

$$Enocde(x) = Encode_{MLM}(x). \tag{6}$$

The output of the encoder is concatenated with other non-textual metadata features as the input of the classifier network. The classifier network is a fully connected feed-forward neural network with one hidden layer activated by ReLU and the output layer activated by sigmoid functions to estimate the probabilities of multiple tags. The tags whose estimated probabilities are greater than or equal to a threshold p_0 are considered the predicted tags of the corresponding column. The range of number of predicted tags is between 0 and the number of supported tags, with both sides inclusive. The number of neurons in the hidden layer is 500. The non-textual metadata features are mainly the column statistical histogram, which describes the distribution of the number of entity cell values in 1,024 lexicographical order buckets. The model only extracts the histogram of a column when the number of distinct cell values are less than or equal to 1,024. Otherwise, the values of all bins are set to 0.

To handle class imbalance and highlight the loss assigned to the misclassified examples, the filtering model uses the α-balanced Focal loss [19] as the objective function for each tag. For the j-th tag in the i-th row in a batch of input, let p_{ij} be the estimated probability, $y_{ij} \in \{0, 1\}$ the ground truth label, $\alpha \in [0, 1]$ a weighting factor, and $\gamma \geq 0$ the focusing parameter, we define $(\alpha_t, p_{t_{ij}}) = \begin{cases} (\alpha, p_{ij}), & \text{if } y_{ij} = 1 \\ (1-\alpha, 1-p_{ij}), & \text{otherwise} \end{cases}$. Suppose the mini-batch size of the filtering model input is N and the number of tags supported is M, the α-balanced Focal loss for multi-label classification can be represented as:

$$\mathcal{L}_{FL}(p_t) = -\frac{1}{N}\sum_{i}^{N}\sum_{j}^{M} \alpha_t (1 - p_{t_{ij}})^{\gamma} log(p_{t_{ij}}).$$

Varying α in the Focal loss function can generate models of various precision and recall pairs, which can be later integrated into the whole framework for performance tuning according to our suggestions in Section 3.2.2.

Overall, let $Classify_{Focal}$ represent the classifier network, according to Formulae 5 and 6, predicting candidate tags of the column c with the filtering model can be represented as:

$$f(c) = Classify_{Focal}(Encode_{MLM}(Embed(\mathcal{M}_c^t)), \mathcal{M}_c^n).$$

3.4 The Verification Models

The implementation of verification model depends on the types of candidate tags. For knowledge-based tags, we check the associated verifiers one by one based on column metadata or content, whereas for ML based tags, we augment the filtering model by incorporating column data features to verify the tags.

3.4.1 Knowledge-based verifiers. If the candidate tag t's verifier only relies on metadata, it is executed only once, i.e., $V_t(\mathcal{M}_c)$. In the case when the verifier uses column content ($\mathcal{D}_c$), we need to check the verifier repeatedly for every cell value. $\mathcal{D}_c$ essentially is an array of cell values each corresponding to a row in the relational table. We accept the tag if the percent of positive verification results for all these cell values is larger than a predefined threshold (e.g., 90%). Null values are directly skipped. For example, if the verifier for t is regular-expression-based and there are 100 rows (cell values) in the column content, we test whether the cell values match the regular expression or not. If 90 cells passed the test, t is accepted as the column's final tag. In practice, such threshold approach is very effective since it can tolerate a small fraction of damaged data and the overly-restrictive regular expression patterns [22].

3.4.2 Machine learning verifier. The machine learning verifier is a multi-label classifier. Its structure (depicted in Figure 2) is similar to that of the filtering model, which also uses multiple pre-trained Transformer layers to encode the textual features. However, there are differences in the embedding layer, the encoder network, the classifier network, and the loss function, respectively.

First, the input of a machine learning verifier includes the column data $\mathcal{D}_c$ as the model input besides the column metadata $\mathcal{M}_c$ of a column c. The column data include the sample entity cells. Different from Formula 5, the output of embedding the column c is:

$$Embed(c) = (Embed(\mathcal{M}_c^t), Embed(\mathcal{D}_c)). \quad (7)$$

Second, the machine learning verifier is further fine-tuned with the Masked Entity Recovery (MER) objective [12] for column textual content besides the Masked Language Model (MLM) objective for column textual metadata. Given a set of entity cells, the MER objective masks some entity cells at random and tries to recover the linked entity based on surrounding entity cells and column metadata. Recall that the MLM objective helps the pre-trained encoder capture the lexical, semantic, and contextual information of texts. The MER objective helps the pre-trained encoder capture the context information about the relationship between the column metadata and the entity cells. Let $Encode_{MM}$ denote such an encoder fine-tuned with both MER and MLM objectives, given a column embedding $x = (x_m, x_e)$, where x_m is the column metadata embedding and x_e is the column content embedding, the output of the encoding x in the verifier is:

$$Enocde(x) = (Encode_{MM}(x_m, x_e)). \quad (8)$$

Third, to handle the increased dimension of the encoder output, the number of hidden layers in the verifier classifier network increases from 1 to 2. The number of neurons in these two hidden layers are both 1000.

Fourth, the verifier network use the multi-label binary cross-entropy (BCE) loss as the objective function. Given the number of labels the model supports M, the mini-batch size N, suppose for the j-th label of the row i, the ground truth is $y_{ij} \in \{0, 1\}$ and the output estimated probability is p_{ij}, we define the multi-label binary cross-entropy loss as:

$$\mathcal{L}_{BCE}(p, y) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}log(p_{ij}) + (1 - y_{ij})log(1 - p_{ij})$$

Overall, let $Classify_{BCE}$ represent the classifier network of the machine learning verifier V_t, according to Formulae 7 and 8, verifying if a candidate tag t is one of the ultimate tags of the column c can be represented as:

$$V_t(c) = Classify_{BCE}$$
$$(Encode_{MM}(Embed(\mathcal{M}_c^t), Embed(\mathcal{D}_c)), \mathcal{M}_c^n).$$

4 IMPLEMENTATION

This section describes our implementation issues of the `T2` framework in the real cloud environment. Section 3 describes the framework from a single column's perspective, but in our implementation, columns of the same table are examined in a batch to improve efficiency.

At the front end, the auto-tagging service can provide an interface for users to choose which instances or databases need to be analyzed for tagging. This way, databases of no interest, e.g., for testing and development, can be excluded from tagging to save costs.

At the backend, the auto-tagging service loops over a list of tables for each database the user choose. Then, for each table i from the database, we run the two-phase tagging procedure for all the columns in the table *in batch mode*. In other words, we perform tagging in the granularity of the table instead of the column. The pseudocode of tagging columns for one table is shown in Algorithm 1. Such batch mode is beneficial because database systems usually provide mechanisms to read table-wise and column-wise metadata for one table within one operation. Columns of the same table can share the table-wise metadata if examined in the same batch, thus preventing redundantly retrieving table-wise metadata and reducing data communication costs For instance, we can run the following SQL query to retrieve basic metadata for all the columns in a table in one run: `SELECT * FROM information_schema.columns WHERE TABLE_SCHEMA=db_name and TABLE_NAME=tb_name`. Specifically, in the first phase of the `T2` framework, the algorithm retrieves the metadata of all the columns in table i (denoted by $\mathcal{M}^i$) and runs the filtering model together for those columns (lines 1-2). The output of the filtering model (represented by $\hat{T}$), combined with the emerging tag set, constitutes the candidate tag set T^{1F} (line 4-6).

In the second phase of `T2`, column data ($\mathcal{D}^i$) are needed only if the candidate tag set is not empty (lines 8-10). In this case, various verifiers will be launched, using column metadata or content as input. Then, the algorithm starts verifying the candidate tags for each column. First, candidate tags for each column (T_c^{1F}) are divided into three categories according to the category of tag verifiers (lines 10-12). KT_c^{1F}, PMT_c^{1F}, and EMT_c^{1F} represent knowledge-based candidate tags, ML-based candidate tags from the published tag set, and ML-based candidate tags from the

Algorithm 1: Column tagging for one table

```
Input: Table $i$, emerging tag set $T^e$.
Output: $T^{2F}$ map of all columns in $i$ to accepted tags.
// Phase 1
1  $\mathcal{M}^i \leftarrow$ column metadata of table $i$ retrieved in batch;
2  $\hat{T} \leftarrow f(\mathcal{M}^i)$;
// Phase 2
3  Initialize $\mathcal{D}^i \leftarrow \emptyset$;
// combine filtering output with emerging tags
4  foreach column $c$ in table $i$ do
5      $T_c^{1F} \leftarrow \hat{T}_c \cup T^e$;
6  end
7  if $T^{1F} \neq \emptyset$ then
8      $\mathcal{D}^i \leftarrow$ column data of table $i$ retrieved in batch;
9  end
10 foreach column $c$ in table $i$ do
11     Separate $T_c^{1F}$ into three sets: $KT_c^{1F}$, $PMT_c^{1F}$ and $EMT_c^{1F}$;
12 end
// check knowledge-based verifiers
13 foreach column $c$ in table $i$ do
14     foreach tag $t$ in $KT_c^{1F}$ do
15         if $V_t(c) == true$ then
               // add knowledge-based tag
16             $T_c^{2F} \leftarrow T_c^{2F} \cup \{t\}$
17         end
18     end
19 end
// check ML-based verifiers
20 $PMT^{2F} \leftarrow V_m(\mathcal{M}^i, \mathcal{D}^i)$;
21 $EMT^{2F} \leftarrow V_m^e(\mathcal{M}^i, \mathcal{D}^i)$;
22 foreach column $c$ in table $i$ do
23     $\mathcal{M}^p \leftarrow PMT_c^{1F} \cap PMT_c^{2F}$;
24     $\mathcal{M}^e \leftarrow EMT_c^{1F} \cap EMT_c^{2F}$;
25     $T_c^{2F} \leftarrow T_c^{2F} \cup \mathcal{M}^p \cup \mathcal{M}^e$;
26 end
27 $T^{2F} \leftarrow$ a map of $c$ to $T_c^{2F}$ for every column $c$ in table $i$;
28 return $T^{2F}$;
```

emerging tag set, respectively. Afterward, we can check the tag verifiers separately. For each knowledge-based tag t, we check its associated verifier (V_t) and accept a tag as the final result if the verifier is evaluated true based on the column's metadata $\mathcal{M}_c^i$ or data content $\mathcal{D}_c^i$ (lines 13-19). For ML-based tags, we run the prediction of the published and emerging tag models separately (lines 20-21) and then accept the candidate tags confirmed by the ML models (lines 22-26). Finally, the algorithm returns a map of columns to their corresponding candidate tag sets that have passed the verification phase (i.e., T_c^{2F}).

5 EVALUATION

This section evaluates T2 framework, and compares it with naive approach of exhaustive checking and the the state-of-the-art approaches like Sherlock [14] and TURL [12]. First, we introduce the experimental settings and datasets. Then, we define three key performance metrics that can reflect whether a tagging algorithm is suitable for the cloud environment. Finally, comparison details with respect to these metrics are described one by one.

Table 4: Summary of the open datasets

Dataset name	Source	Number of tables	Number of tags
WikiTables-train	TURL	397,098	255
WikiTables-test	TURL	4,764	248
parent_tables (reduced)	GitTables	2,119	–
real_time_tables (reduced)	GitTables	11,010	–

Table 5: Mixed datasets with different ratios of tables without any tags

Dataset name	Mixed datasets	Number of tables	Ratio of tables without tags
Mix_Wp	WikiTables-test & parent_tables	6,883	30.8%
Mix_Wr	WikiTables-test & real_time_tables	15,774	69.8%
Mix_Wpr	WikiTables-test & parent_tables & real_time_tables	17,893	73.4%
Mix_pr	parent_tables& real_time_tables	13,129	100%

5.1 Settings and Datasets

All the experiments of the paper are conducted on a Linux server with Intel 13th Gen i7-13700K CPU, NVIDIA GeForce RTX 4080 GPU and 64 GB RAM.

To make the comparison fair, we use the same WikiTable dataset as TURL for training, which is publicly available at [12] and contains 397,098 tables and 255 tags. The difference, however, lies in the prediction part, where we import the dataset into MySQL of version 8.0.34 in order to mimic the real production environment where data exist in user databases rather than in a standalone file. This way, we can consider the end-to-end cost of auto-tagging algorithms, including time for creating/closing database connections, reading metadata and content from databases, and running the tagging models. To circumvent the constraint that MySQL disallows identifiers to contain special characters, we convert page titles/captions and column names in the original dataset to MySQL table comments and column comments, respectively.

To assess the effectiveness of T2's filtering model on unrelated tags, we introduce another two open datasets, real_time_tables and parent_tables from the GitTables repository [15] for testing. We discard tables whose column tags are suspected to contain at least one of the above 255 training tags by considering the textual similarity of tag names. As a result, we get two reduced datasets with 11,010 and 2,119 tables, respectively, whose columns are deemed to have no supported tags as the ground truth. By mixing these reduced datasets with the WikiTables dataset in different combinations, we can create real-world scenarios where only a portion of columns from user databases have semantic tags. Details of each single and mixed dataset are summarized in Tables 4 and 5, respectively. Since we train the T2 models only on the WikiTables dataset, the 255 tags from the WikiTables dataset constitute the published tag set, which will be used for prediction on other datasets.

Regarding the training procedure of the filtering model and the machine learning verifier, we first initialize the network weights of their embedding layer and encoder using the same pre-trained checkpoint as TURL [12], which is pre-trained on an unlabeled Wikipedia table corpus [8]. We then fine-tune all the network weights across the entire model using WikiTables training dataset for 10 epochs. In each epoch, the model iteratively processes each batch of data, calculating the loss and performing backpropagation for each batch to update the network's weights.

5.2 Baselines and Metrics

We consider the following four baselines for performance comparison.

Sherlock [14]: It trains a deep neural network with global and character-level statistical features, words embeddings and paragraph embeddings. However, it does not utilize pre-trained networks, compared with TURL. The embedding networks for textual content including words and paragraphs are trained jointly with other features with random initialization.

TRUL [12]: It is an end-to-end deep neural network utilizing multiple pre-trained Transformer layers to encode textual column content. The pre-trained encoder captures not only the lexical, semantic, contextual information of words, but also the knowledge of associations between table content and metadata. The network is fine-tuned on the training dataset of WikiTables with a cross-entropy loss.

T2 with Random Forest (RF) Model: The verifying models of T2 are all replace with the Random Forest, a traditional ML model. For each of the 255 tags in the TURL dataset, we train a random forest model, which serves as the verifier for that tag.

Exhaustive Check of All Models (EC): This baseline is a naive approach that exhaustively checks every tag's verifying model without going through the filtering phase to find all possible tags for every column.

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

- **Ratio of scanned tables**: This metric is defined as the total number of table scan operations (for reading table content) executed by the tagging algorithm during prediction over the number of tables in the dataset. It not only measures every approach's intrusiveness to user databases but also reflects the effectiveness of our filtering model. If the filtering model can filter out unrelated tags, table scan operations required by the verification phase can be saved.
- **Execution Time**: It is the overall time cost that an algorithm pays to finish the tagging task for all the columns in the dataset. We consider the end-to-end time which involves the time for creating/closing database connections, loading metadata and content from databases, and the prediction time. Therefore, it can lead to a fair comparison in realistic environment.

Table 6: Tagging performance comparison of different auto-tagging approaches

Model	Precision	Recall	F1
Sherlock [14]	0.888	0.764	0.822
TURL [12]	0.949	0.945	0.947
EC(RF)	0.916	0.870	0.893
EC(DL-KB)	0.955	0.953	**0.954**
$\alpha = 0.5$ in Focal Loss			
Filter	0.908	0.910	0.909
T2(Filter+RF)	0.951	0.840	0.893
T2(Filter+DL-KB)	0.961	0.937	0.948
$\alpha = 0.99$ in Focal Loss			
Filter	0.633	0.985	0.771
T2(Filter+RF)	0.921	0.868	0.894
T2(Filter+DL-KB)	0.958	0.950	**0.954**

5.3 Tagging Performance

To compare tagging performance, we run all tagging algorithms on the same dataset, i.e., WikiTables-test in Table 4. The results of the precision, recall, and $F1$ score of different models are listed in Table 6, where model names and meanings are described as follows.

- EC(RF): EC with every verifier using the Random Forest model.
- EC(DL-KB): EC with verifiers using the combination of deep pre-trained language models and knowledge-based models (DL-KB).
- Filter: The filtering model in T2, whose performance varies with different values of α in the Focal loss.
- T2(Filter+RF): T2 with the filtering model and Random-Forest-based verifiers.
- T2(Filter+DL-KB): T2 with the filtering model and DL-KB verifiers.

In section 3, we describe T2's verification phase leverage deep pre-trained language models and knowledge-based models (DL-KB) according to the types of candidate tags. Here, we try to prove the advantage of this approach by comparing its performance with the counterpart with Random Forest models.

First, when compared with EC, T2 experiences a slight reduction in recall, because it incorporates a filtering model which will predict some true tags to be false (false negative) if its recall does not reach 1.0. However, on the flip side, the filtering model has already filtered out some false tags, preventing them from entering the verification phase, which in turn improves overall precision. Consequently, T2 exhibits a reduced recall but increased precision, resulting in an $F1$ score that closely aligns with EC. This phenomenon also hold in both cases of comparing T2(Filter+RF) with EC(RF) and comparing T2(Filter+DL-KB) with EC(DL-KB).

Higher recall of the filtering model can reduce overall recall loss, and the improvement in precision of the filtering model can also enhance overall precision. This leads to a trade-off problem: should the filtering model aim for higher recall at the expense of lower precision, or should it strive for a balance between precision and recall. Figure 3 shows the impact of the filtering model's precision and recall on the overall performance. In the comparison, we find that achieving higher recall with the filtering model is preferable improve the $F1$ score on the WikiTables dataset. The

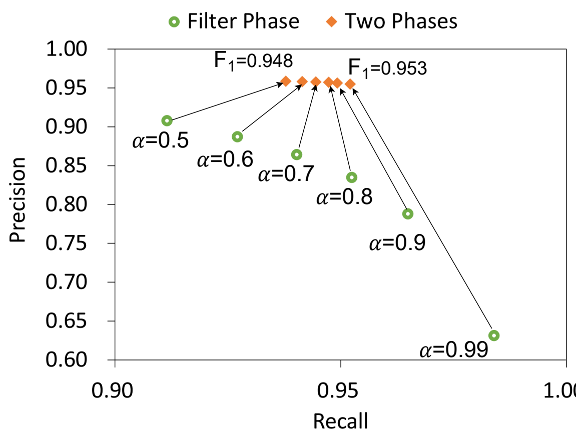

Figure 3: Impact of the filtering model performance on the overall performance, by varying α in the Focal loss

Table 7: Tagging performance of Filter and T2 without histogram feature

Model	Precision	Recall	F1
α = 0.5 in Focal Loss			
Filter	0.899	0.907	0.903
T2(Filter+RF)	0.947	0.840	0.890
T2(Filter+DL-KB)	0.957	0.929	0.943
α = 0.99 in Focal Loss			
Filter	0.596	0.984	0.743
T2(Filter+RF)	0.921	0.868	0.894
T2(Filter+DL-KB)	0.955	0.945	**0.950**

precision and recall of filtering model can be adjusted by varying the value of α in the Focal loss function. As an illustration, when $\alpha = 0.5$, the filtering model maintains a balance between recall and precision, and when $\alpha = 0.99$, the filtering model significantly increases recall but reduces precision, resulting in the overall $F1$ scores of 0.948 and 0.954 for T2(Filter+DL-KB), respectively, both of which outperform the state-of-the-art TURL model.

Utilizing the pre-trained language model significantly improves the tagging performance. Sherlock's performance is reasonably worse compared to other models as it merely utilizes column content and does not rely on metadata. In contrast, our filtering model purely relies on metadata but can outperform Sherlock when the value of α in Focal Loss is adjusted to 0.5. This clearly demonstrates the importance of metadata and the advantage of using a pre-trained language model that embeds semantic and contextual information.

Furthermore, considering the scenario where some user databases miss statistics metadata like histogram, we also conducted experiments with histogram features removed. Table 7 shows the performance results. Comparing Table 7 and Table 6, we can observe that removing histogram only causes slight performance drop in all cases. Specifically, the best $F1$ score only decreases from 0.954 to 0.950.

5.4 Intrusiveness to Database

Figure 4 depicts the ratio of scanned tables by T2 and baseline approaches when tested on different datasets. It is not surprised to find that this metric is 100% for EC, Sherlock and TURL in all datasets, because they all rely on table content and have to scan every table, no matter what dataset is used. In comparison, T2 has lower generally ratios, saving expensive table scan operations. For example, when T2 is executed on Mix_Wr dataset, only 45.6% of tables need to be scanned. The rest of the tables are predicted by the filtering model to be irrelevant to any of the published tags. In the best case, T2 can save 73.5% table scan operations on Mix_pr dataset. As a result, T2 can greatly mitigate the adverse impacts of the auto-tagging algorithm on user databases, achieve lower intrusiveness.

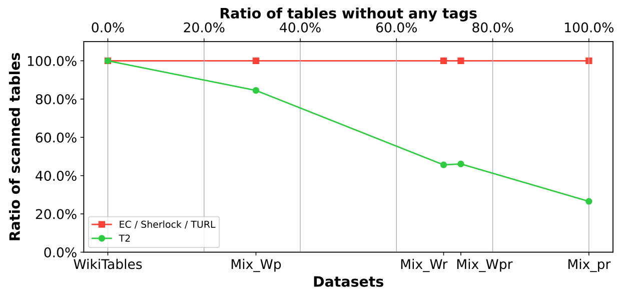

Figure 4: Ratio of scanned tables by different auto-tagging approaches with datasets of different ratios of tables without any tag

As observed in Figure 4, the ratio of scanned tables in T2decreases nearly linearly as the ratio of tables without any tags in the dataset increases. This phenomenon implies that T2 will perform much better when only a small portion of user data should be annotated. In reality, we have found that such situation is the norm because enterprises are commonly concerned about a small portion of essential/sensitive data being protected or cataloged. Therefore, T2 only incurs limited impact on user databases in real environment and is more likely to be accepted by customers compared with other approaches.

5.5 Execution Time

The execution time for T2 and baseline approaches are shown in Figure 5. Each bar in the figure is the end-to-end execution time for one approach on one dataset. To discover the execution bottleneck, we breakdown the execution time for individual operations (by different textures in the bar), including loading metadata, loading column content, prediction time, T2 phase 1 filtering, and T2 phase 2 verification.

From the figure, we can see that the execution time of T2 is longer than TURL on WikiTables dataset. This is because when all the table has at least a column with tags, every table needs go through the filter and verification phase in T2. However, as the number of tables with no tags for their columns increases, T2begins to show its advantage. For all datasets Mix_Wr, Mix_Wpr and Mix_pr with a specific percentage of tables without any tags, T2 outperforms all the baseline approaches. Specifically, on the Mix_pr dataset, T2 reduces the execution time by 25.2% compared with TURL. This performance gain mainly comes from the reduction of time in loading column content contributed the filtering model. For EC, its execution time performance is the worst, because it not only requires loading metadata and column content, but also needs to verify every tag.

Moreover, we study the average execution time for each column (depicted in Figure 6), and find that it decreases as the ratio

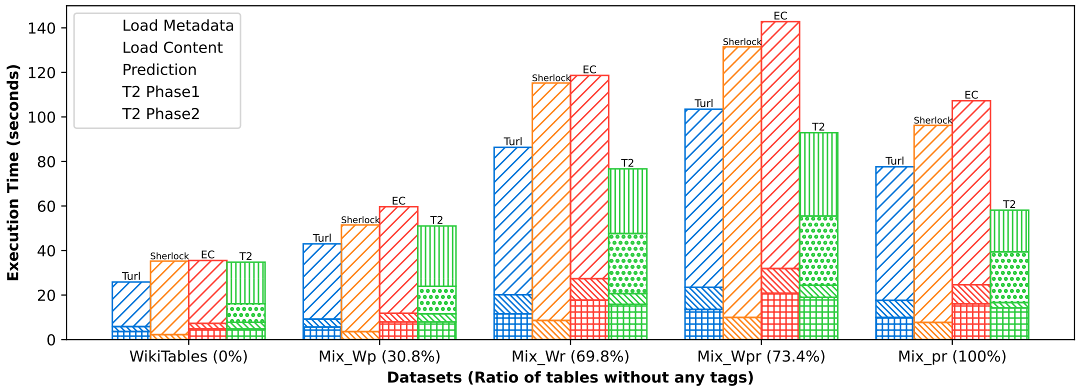

Figure 5: End-to-end execution time for different auto-tagging approaches.

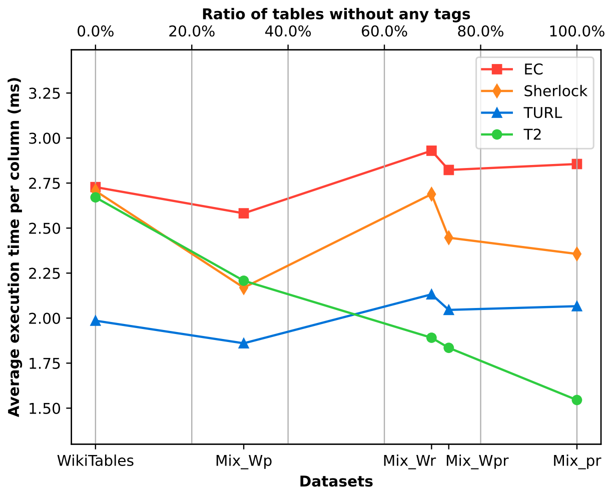

Figure 6: Execution time per column for different auto-tagging approaches.

of tables without any tags grows in T2. T2 outperforms TURL starting from certain point around 50% of tables without any tags in the dataset. This result is consistent with our finding in the previous subsection, and implies that T2 is particularly suitable for scenarios where only a small portion of columns need to be tagged.

6 RELATED WORK

Auto-tagging techniques, which aim to automatically discover semantic types (or tags) from data, are crucial to modern data management, especially in the big data era. Over the past decades, there have been a lot of endeavors from industry and academia to solve auto-tagging problems through various algorithms.

The first class of prior work focuses on regular expression methods. [13, 21] work on efficient methods to construct/learn regular expressions to discover patterns in a table column. XSYSTEM [16] 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 [22] prunes a fraction of data to tolerate non-conforming values to increase the prediction recall. However, regular expression based approaches intrinsically need to read and examine table content, thus can lead to high computation cost and impact on user databases.

Another body of work tries to relate table columns to tags via the value overlapping method [11, 25, 26] or the synthesis method [27]. The former method selects tags for a target table column based on how its content overlaps with other columns in a knowledge base. It usually needs a full scan of the table column to improve accuracy and relies on identical matching which fails to consider the contextual and semantics information of the text content. The latter method searches for or generates validation functions to verify if table column is matched to some tags. These synthesized functions usually incorporate domain knowledge and are only applicable to specific types of tags whose content obey certain protocols or standards, e.g., ISBN and credit card numbers. On the contrary, we endeavor to develop generic approaches to support different types of tags from diverse customers. In fact, all the above tag specific methods are orthogonal to our approach, and can be reused as verifiers in the second phase of T2 famework.

In recent years, machine learning approaches for table column tagging have become a trend. Sherlock [14] trains a deep neural network with abundant features, including global and character-level statistical features, word embedding, and paragraph embeddings. Sato [28] includes table context and relationships among neighboring columns in the learning model. Some studies [29, 30] use pre-trained language models to leverage the contextual semantics learned from large corpora to significantly improve the accuracy with less training effort. TURL [12] fine-tunes the pre-trained Transformer-based network and learns column semantic types from table contextual information. DODUO [23] extends the work by adopting multi-task learning into the learning model to share the knowledge learned from column semantic types and relation tasks. T2 also leverages pre-trained Transformer-based language models in both the filtering and verification phase of the framework. The main drawback of existing models is that they are end-to-end learning models, which lack the flexibility to utilize domain knowledge for annotating domain-related tags and to support emerging tags raised ad hoc by cloud customers.

Essentially, T2 is a hybrid approach mixing knowledge based and machine learning based models. In the literature, there also exist some hybrid learning models that incorporate domain knowledge. Chen et al. [10] 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 [9] ensembles the results from a convolutional neural network and a knowledge-based look-up model to determine the final column semantic types. T2 differs from them in that it applies the filtering model and verification model in sequence instead of in parallel. The advantage of running model sequentially is that columns with no tags can be identified at the filter phase, and thus be skipped for verification. As a result, it reduces the tagging cost and runs more efficiently. Moreover, the verifiers of T2 are not limited to look-up models.

7 CONCLUSION AND FUTURE WORK

In this paper, we rethink the design of auto-tagging algorithms from a cloud provider's perspective based on our observation that tagging accuracy, efficiency, low intrusiveness to user databases, and extensibility must be considered simultaneously in the cloud environment. To this end, we propose a novel two-phase auto-tagging framework, T2, consisting of the filter phase and verification phases. T2 is efficient and poses lightweight impacts on user databases because it can identify irrelevant tags in the filter phase and skip scanning user databases to verify them. Furthermore, its hybrid architecture achieves high tagging accuracy by leveraging both domain knowledge and deep pre-trained language models. Existing tagging algorithms that domain experts have created can be easily plugged into the framework as tag verifiers in the second phase. Overall, the novel design enables T2 to outperform the state-of-the-art approaches in various aspects.

In the future, we plan to explore in two directions. First, we will continuously enrich the supported knowledge-based verifiers to cover more domains. Second, auto-tagging algorithms usually run on a regular basis to refresh the tagging result; we will try to reduce the frequency of executing T2 and make it more efficient, especially when the schema or content of user databases has not/slightly been changed.

REFERENCES

[1] 2023. *Alibaba Cloud Data Security Center.* https://www.alibabacloud.com/product/sddp

[2] 2023. *AWS Glue.* https://aws.amazon.com/glue

[3] 2023. *Google Cloud Dataplex.* https://cloud.google.com/dataplex

[4] 2023. *Microsoft Purview.* https://azure.microsoft.com/products/purview

[5] 2023. *MySQL ANALYZE TABLE.* https://dev.mysql.com/doc/refman/8.0/en/analyze-table.html

[6] 2023. *SparkSQL ANALYZE TABLE.* https://spark.apache.org/docs/latest/sql-ref-syntax-aux-analyze-table.html

[7] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. 2015. Profiling relational data: a survey. *The VLDB Journal* 24, 4 (Aug. 2015), 557–581.

[8] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. 2015. Tabel: Entity linking in web tables. In *International Semantic Web Conference.* Springer, 425–441.

[9] 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.

[10] 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.

[11] 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.

[12] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. TURL: table understanding through representation learning. *Proceedings of the VLDB Endowment* 14, 3 (2020), 307–319.

[13] 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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision.* 2980–2988.

[20] Masayo Ota, Heiko Müller, Juliana Freire, and Divesh Srivastava. 2020. Data-Driven Domain Discovery for Structured Datasets. In *Proceedings of the VLDB Endowment.* 953–965.

[21] 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.

[22] 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.

[23] 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.

[24] Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Hao Tian, Hua Wu, and Haifeng Wang. 2020. Ernie 2.0: A continual pre-training framework for language understanding. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 8968–8975.

[25] 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).

[26] 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.

[27] 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.

[28] Dan Zhang, Madelon Hulsebos, Yoshihiko Suhara, Çağatay Demiralp, Jinfeng Li, and Wang-Chiew Tan. 2020. Sato: contextual semantic type detection in tables. *Proceedings of the VLDB Endowment* 13, 12 (2020), 1835–1848.

[29] Ziqi Zhang. 2017. Effective and efficient semantic table interpretation using tableminer+. *Semantic Web* 8, 6 (2017), 921–957.

[30] 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.