Tab-Cleaner: Weakly Supervised Tabular Data Cleaning via Pre-training for E-commerce Catalog

Kewei Cheng[1], Xian Li[2], Zhengyang Wang[2], Chenwei Zhang[2], Binxuan Huang[2]

Yifan Ethan Xu[3], Xin Luna Dong[3] and Yizhou Sun[1]

[1]University of California, Los Angeles [2]Amazon, [3]Meta AI

{viviancheng,yzsun}@cs.ucla.edu,
{xianlee, zhengywa, cwzhang, binxuan}@amazon.com
Ethan.yifanxu@gmail.com, lunadong@fb.com

Abstract

Product catalogs, conceptually in the form of text-rich tables, are self-reported by individual retailers and thus inevitably contain noisy facts. Verifying such textual attributes in product catalogs is essential to improve their reliability. However, popular methods for processing free-text content, such as pre-trained language models, are not particularly effective on structured tabular data since they are typically trained on free-form natural language texts. In this paper, we present Tab-Cleaner, a model designed to handle error detection over text-rich tabular data following a pre-training / fine-tuning paradigm. We train Tab-Cleaner on a real-world Amazon Product Catalog table w.r.t millions of products and show improvements over state-of-the-art methods by 16% on PR AUC over attribute applicability classification task and by 11% on PR AUC over attribute value validation task.

1 Introduction

Product catalogs are widely used by E-commerce websites to organize product information (Dong et al., 2020). They can be conceptualized as wide tables where each row corresponds to a product and each column corresponds to an attribute (Table 1). Most of the product catalog data are self-reported by individual retailers and thus inevitably contain various types of errors (Dong et al., 2020). It is critical to clean the data to avoid cascading errors harming downstream applications (Pujara et al., 2017; Chu et al., 2016).

Due to product catalogs' wide tabular format and rich textual content, attribute cleaning poses a number of challenges as we outline below.

C1: Product catalogs are structured tables with unstructured textual values. Errors in product catalogs are indicated by column-wise, row-wise, and table-wise inconsistencies (Table 1). While common pre-trained language models (LMs) (Rajpurkar et al., 2016; Clark et al., 2020; Beltagy et al., 2019; Martin et al., 2019) are effective in processing free texts, they are not suited to capture tabular structures. Although recent works (Yin et al., 2020; Herzig et al., 2020) have adapted Transformers to jointly query tabular and textual data , their goal is information extraction (e.g. answering SQL/free text questions) or tabular structure prediction, rather than error detection.

C2: Attributes in product catalogs are strongly correlated with each other. For example, in Table 1, *"Cheddar"* is a valid flavor on its own, but contradicts its ingredient column *"Cayenne Pepper, Paprika Extract, Dehydrated Spices"*. Such correlation renders anomaly detection methods focusing only on value distributions in a single column ineffective.

C3: Product catalogs are extraordinarily wide. Considering that some attributes of product catalogs may be text-heavy (e.g., product titles and descriptions are often very long.), a super-long sequence will arise from concatenating all textual attributes of a specific product. Existing table representation models (Yin et al., 2020; Du et al., 2021) restrict input sequence length to a certain budget (e.g., 512) by truncation, which will inevitably cause information loss.

To address the above challenges, we present Tab-Cleaner, a transformer model with a hierarchical-attention mechanism and trained with the pre-training / finetuning paradigm to facilitate data cleaning over text-rich tabular data for E-commerce catalog. Our proposed model is generic. It applies not only to the product domain but also excels in other domains which involve text-rich tabular data. In summary, this paper makes the following contributions.

- We propose a tabular structure-aware pre-training / fine-tuning paradigm to enable a Transformer-based model to process text-rich tabular data.
  - We propose a novel hierarchical attention

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics
Volume 5: Industry Track, pages 172–185

	Product Title	Product Category	Flavor	Ingredient	Color	Size
$\mathbf{R_1}$	Brand A Tortilla Chips † Spicy Queso, 6 - 2 oz bags	chips-and-crisps	Spicy Queso	Ground Corn, Chipotle Pepper Powder, Paprika Extract, Spices	-	6 - 2 oz bags
$\mathbf{R_2}$	Brand B Bean Chips Spicy Queso,† High Protein and Fiber, Gluten Free, Vegan Snack, 5.5 Ounce (Pack of 6)	chips-and-crisps	**Cheddar**	Navy Beans, Cayenne Pepper, Paprika Extract, Dehydrated Spices	-	5.5 Ounce (Pack of 6)
$\mathbf{R_3}$	Brand C Organic Honey, Blossom, 17.6 Ounce†	honey	Blossom	100% pure raw honey straight from the hive	-	17.6 Ounce
$\mathbf{R_4}$	Brand D BPA Free No Spill † Sippy Cup, Orange (9 ounce)	baby-drinkware	**Orange**	-	Orange	9 ounce
$\mathbf{R_5}$	Brand F Women's Spa Studio Green Tea Eye Pads 2 Pack- Each Contains† 5 treatment (Total 10 Treatments)	**green-teas**	Green Tea	Aloe, Camellia Sinensis Leaf Extract, Panax Ginseng Root Extract	Green	Total 10 Treatments

† We mask the brand of the products to avoid revealing sensitive information.

Table 1: An example product catalog, where each row corresponds to a product and each column corresponds to an attribute. A vast majority of attributes in product catalogs are textual attributes. The incorrect attributes are highlighted in red.

mechanism to capture attribute-level correlation.

  - The hierarchical attention also enables a sparse attention pattern to reduce memory consumption and speed-up training, which allows us to cope with long sequences.

- We train Tab-Cleaner on a real-world Amazon Product Catalog w.r.t millions of products and show that we can improve over SOTA methods by 16% on PR AUC over attribute applicability classification task and by 11% on PR AUC over attribute value validation task.

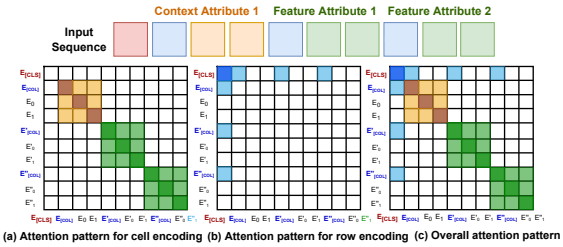

Figure 1: Hierarchical attention mechanism for capturing the interactions among different attributes in a table using a two-level architecture. (a) encodes cells (i.e., attributes) on the basis of their tokens; (b) encodes rows (i.e., products) on the basis of all their cells; (c) the combined hierarchical attention.

2 Problem Definition

Given a product catalog table T, each row corresponds to a product (p_i) and each column corresponds to an attribute (a_j). Cell T_{ij} is the value of attribute j of product i containing a list of tokens. Attributes in product catalog data can be broadly divided into two classes:

- **Context attributes** (A_{context}), which are usually long texts that describe general information of a product (e.g., title, product description).
- **Feature attributes** (A_{feature}), which are usually short texts that describe a specific attribute about a product (e.g., color, size, flavor, scent).

We formally define the problem of data cleaning over the product catalog table as follows:

Given: a product catalog table T,

Identify: incorrect cells about feature attributes $\{T_{ij}\}_{p_i \in P, a_j \in A_{\text{feature}}}$

3 Tab-Cleaner Framework

Since manual annotation of error data is costly and labor-intensive to obtain on E-commerce websites, we follow the pre-training/finetuning paradigm to alleviate the need for large-scale labeled data for data cleaning. Tab-Cleaner is first pre-trained on an unlabeled product catalog corpus with tabular structure-aware pre-training objectives carefully designed to capture the tabular structure. Then, Tab-Cleaner fine-tunes the model using manually curated labeled data.

3.1 Tab-Cleaner Architecture

Transformer-based models cannot be directly applied to tabular data. To flatten each row in the input table into a sequence, we first prepend each attribute value with a [COL] token and its column name, then concatenate them into a flat sequence. For example, Tab-Cleaner flattens R_1 in Table 1 as follows.

[CLS] [COL] Product title: Brand A Tortilla Chips Spicy Queso,6 - 2 oz bags [COL] Product Category: chips-and-crisps [COL] Flavor: Spicy Queso [COL] Ingredient: Ground Corn, Chipotle Pepper Powder, Paprika Extract, Spices [COL] Size: 6 - 2 oz bags

The input structure is designed to capture both attribute and product representations.

Attribute (Cell-level) Representation: The first token of every cell is always a special token [COL]. The final hidden state corresponding to token [COL] is used to represent a cell.

Product (Row-level) Representation: Each

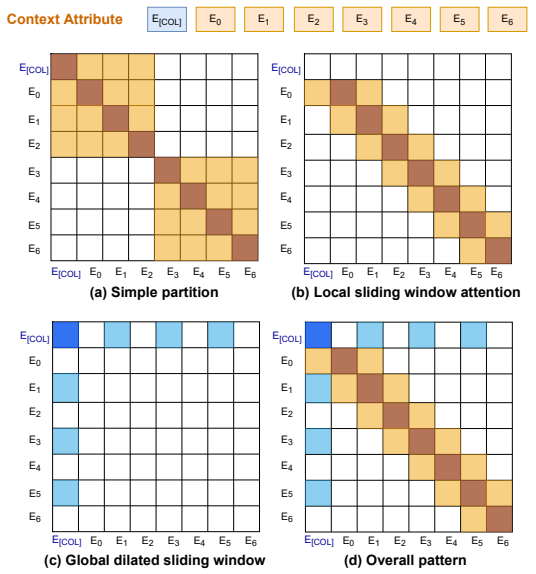

Figure 2: The configuration of attention patterns for context attribute learning. (a) partitions the long context into smaller sequences; (b) local sliding window attention ($w = 3$); (c) global dilated sliding window attention ($d = 2$); (d) the combined model.

row in the product catalog corresponds to a specific product. The first token of a row (i.e.,[CLS]) is used to represent a product.

Tab-Cleaner is implemented by extending DistillBERT's architecture (Sanh et al., 2019) with additional embeddings to capture tabular structure. The detailed architecture is given in Appendix A.

3.2 Hierarchical Attention Mechanism

Long sequences from concatenated product attributes are challenging for Transformer-based models to process due to quadratic scaling in the full self-attention operation. However, full attention to the entire content is not necessary for modeling structured tables. Based on this insight, we propose a hierarchical attention mechanism that models the tabular structure through a two-level architecture, first encoding all cells on the basis of their tokens (Fig. 1 (a)), then encoding the entire rows on the basis of all their cells (Fig. 1 (b)). In this way, we avoid full attention calculation, thereby greatly reducing the memory and computation in need. The detailed implementation is given in Appendix B.

3.2.1 Cell Encoding

A cell is the smallest unit to form a table. The list of tokens within each cell expresses semantics independently of the rest content in a row. A local window that covers only the target cell is enough to learn its semantics. Motivated by such observation, our local attention pattern employs a flexible-size window to include only the tokens of the target cell to calculate its representation as shown in Fig. 1 (a). Attributes in a product catalog fall into two classes: (1) feature attributes, which are usually short and can be easily covered by a small window; (2) context attributes, which can contain thousands of tokens. A window with a limited size cannot cover a context attribute.

Context Attribute Representation Learning A straightforward solution may partition a context attribute into smaller sequences (Fig. 6 (a)). Such partitioning could result in information loss. To address this issue, we propose a novel *local + global attention* to learn the context attribute representation (Fig. 6 (d)).

Local Attention Most information about a token can be derived from its surrounding tokens. We define a sliding window attention to capture local information around each token. Given a fixed window size w, each token attends to $1/2w$ its local neighboring tokens on each side (Fig. 6 (b)).

Global Attention Although the local attention shows great effectiveness in capturing local context as demonstrated in Longformer (Beltagy et al., 2020)), it cannot aggregate the global information into the token [COL]. The [COL] has to attend all tokens across the cell to collect the global information. To reduce the computational cost, we propose a "dilated attention" on [COL] where the window has gaps of size dilation d (Fig. 6 (c)). Note that the "dilated attention" operation is symmetric. All tokens attended by [COL] also attend [COL] tokens. Assuming we set dilation d equal to the window size w. Given a sequence with length as L, we can learn [COL] by attending only ceil$(\sqrt{L})$ tokens.

We discussed the expressiveness of *local + global attention* in Appendix C and showed that it is as expressive as full attention.

3.2.2 Row Encoding

Attributes of products are usually correlated with each other, which is useful to identify incorrect attribute values. For example, R_1 in Table 1 indicates a strong correlation between the ingredient "pepper" and the flavor "spicy". To capture the underlying correlations among attributes, the hierarchical attention mechanism focuses on learning the attention among [COL] tokens (i.e., attribute representation) and [CLS] tokens (i.e., product rep-

(a) Original Table

	Flavor	Color
R_1	Spicy Queso	-
R_5	Green Tea	Green

(b) Swap Cells on the Same Row

	Flavor	Color
R_1	Spicy Queso	-
R_5	**Green**	**Green Tea**

(c) Swap Cells on the Same Column

	Flavor	Color
R_1	**Green Tea**	-
R_5	**Spicy Queso**	Green

Table 2: The different cell corruption strategies. We highlight the swapped attributes in red.

resentation) at its second level as shown in Fig. 1 (b). The learned attention cannot only capture the interaction among attributes but also aggregate the entire content of a row into the special token [CLS].

3.2.3 Conditional Encodings of Feature Attributes over Context Attributes

Context attributes contain useful information about products for verifying the correctness of feature attributes. However, given the hierarchical attention mechanism, feature attributes are learned independently from context attributes. To enable conditional encodings of feature attributes over context attributes, we further improve cell encoding for feature attributes as discussed in Appendix D.

3.3 Pre-training Objectives

In order to pre-train Tab-Cleaner using unlabeled product catalog tabular corpus, we adopt the Masked Language Model (MLM) objective for learning token-level representations. In addition, we also propose several different objectives for tabular structure representation learning (e.g., cell-level and row-level representations).

Objective for learning token level representations: We apply the standard Masked Language Modeling (MLM) objective to learn token-level representations, with a masking rate of 15%. Since MLM lacks the ability to decompose the tabular structure, we also propose two different objectives for tabular structure representation learning:

Objective for learning cell-level representation: Essentially, we corrupt a certain percentage of cells and then learn a classifier to decide if the cell has been corrupted. This objective enables the model to identify incorrect attributes. We use two different corruption strategies to generate corrupted cells as shown in Table 2.

- **Swap cells on the same row:** randomly swap two attributes of the same product, e.g. switch the attribute value of color and flavor of R_5 to construct corrupted cells (Table 2(b)).
- **Swap cells on the same column:** randomly swap an attribute of a product with the same attribute from another product, e.g. switch the flavor attributes of R_1 and R_5 (Table 2(c)).

A binary classifier is placed over the final hidden state corresponding to the token [COL] to decide whether the cell has been corrupted.

Objective for learning row-level representation: Each product in the product catalog is associated with a label indicating its category. To learn row-level representation, we apply a multi-class classifier over the final hidden state corresponding to [CLS] token to predict the category of the product. This objective helps the model to understand the entire content of a product.

Both objectives for learning cell and row level representation can be modeled using cross-entropy between the one-hot label and the prediction:

$$\mathcal{L} = \sum_k y_k \log p_k \tag{1}$$

where y_k is the true label and p_k is the softmax probability for the k-th class. The final objective function is formulated by combining all three objectives together.

3.4 Fine-tuning

The pre-training procedure is followed by the fine-tuning stage on labeled data. During the fine-tuning stage, we apply Tab-Cleaner to identify two kinds of data errors:

- **Inapplicable attribute**, which refers to an attribute that a product should not have. For example, a sippy cup is not edible and thus should not have the attribute "flavor" (R_4 in Table 1);
- **Incorrect attribute value**, which refers to incorrect value of an attribute. For example, the category of product *"Women's Spa Studio Green Tea Eye Pads"* should be *"eye pad"* instead of *"green teas"*. (R_5 in Table 1).

To predict the correctness of an attribute, its representation ([COL] embeddings) is fed into a two-layer network with ReLU activations. The output is then used to predict the correctness from a sigmoid layer, training with a binary classification objective.

We demonstrate that Tab-Cleaner is effective in detecting both inapplicable attributes and incorrect attribute values in experimental studies.

4 Experiments

In this section, we evaluate Tab-Cleaner over two different data cleaning downstream tasks on real-world Amazon datasets.

4.1 Datasets

Datasets for Pre-training We construct two tabular corpora based on the product data obtained from the public Amazon website for pre-training. Due to the different numbers of attributes included, we call these two pre-training tables standard table and wide table. Specifically, the wide table is constructed to investigate how Tab-Cleaner deals with extremely long sequences. Detailed information has been introduced in Appendix E.1 and E.2.

Datasets for Fine-tuning To ascertain the performance of Tab-Cleaner, we study two downstream tasks: attribute applicability classification and attribute value validation. Details are provided in Appendix E.2.

4.2 Experimental Setup

Pre-training & Fine-tuning We train Tab-Cleaner for three epochs for pre-training and 10 epochs for fine-tuning. Detailed settings are in Appendix E.3.

Evaluation Metric. Our goal is to identify incorrect attributes of a product, which is a binary classification problem. We adopt the area under the Precision-Recall curve (PR AUC), the area under the Receiver Operating Characteristic Curve (ROC AUC), and Recall at Precision=X (R@P=X) for evaluation. Details about these metrics are given in Appendix E.3.

Compared Methods. We evaluate Tab-Cleaner against state-of-the-art (SOTA) algorithms, including (1) DistillBERT (Sanh et al., 2019), since Tab-Cleaner is implemented by extending DistillBERT; (2) Transformer for Longer Sequences (e.g., Longformer (Beltagy et al., 2020)); (3) Nature Language Inference method (NLI). Details about the baseline methods are given in Appendix E.3. We did not include tabular representation models as our baseline because they cannot be applied to our scenario.

4.3 Data Cleaning Tasks

Attribute Applicability Classification We require each method to predict the applicability of the attribute in the test dataset. Details are provided in Appendix E.4. As presented in Table 3: (1) NLI performs the worst among all methods, indicating the necessity of jointly leveraging all attributes to detect error; (2) Tab-Cleaner consistently outperforms baselines in all cases with significant performance gain (improving SOTA from 0.296 to 0.379 on R@P=0.9).

Attribute Value Validation We require each method to validate the correctness of the attribute value in the test dataset. As shown in Table 4, TabCleaner handles both short sequences and long sequences very well.

4.4 Scalability

To demonstrate the scalability of Tab-Cleaner, we present training time and memory cost for pre-training over the wide table in Table 5. Specifically, we train Tab-Cleaner for three epochs where Tab-Cleaner has 6 layers, a hidden dimensionality of 768, 12 heads, and a batch size of 32. To fairly compare different transformer-based methods, the same setting is employed for all models. Before pre-training, we truncate each row's contents to 512 tokens to make training feasible for DistillBERT. Tab-Cleaner shows the best performance in terms of both pre-training time and memory cost. The superiority of Tab-Cleaner can ascribe to the sparse attention pattern enabled by the hierarchical-attention mechanism.

Methods	Pre-training Time (Hours/Epoch)	Memory Cost (MB)
DistillBERT	26.95	31263
Longformer	60.56	32391
Tab-Cleaner	**21.63**	**26501**

Table 5: Training time and memory cost of different methods.

4.5 Ablation Study

Improvement brought by pre-training Tab-Cleaner follows the pre-training/finetuning paradigm to alleviate the need for large-scale labeled data for data cleaning. To validate the improvement brought by pre-training, we derive a baseline Tab-Cleaner without pre-training and compare it with Tab-Cleaner over attribute applicability classification task as shown in Fig. 3. Tab-Cleaner without pre-training directly fine-tunes over the distilled version of the BERT base model without pre-training over the Amazon product catalog corpus. We observe that the pre-training brings significant performance gain: Tab-Cleaner with pre-training increases the PR

Pre-training over Standard Table						
Methods	PR AUC	ROC AUC	R@P=0.6	R@P=0.7	R@P=0.8	R@P=0.9
NLI	0.33	0.832	0.237	0.181	0.11	0.079
DistillBERT	0.593	0.907	0.561	0.47	0.411	0.296
Longformer	0.554	0.905	0.501	0.462	0.395	0.245
Tab-Cleaner	**0.613**	**0.91**	**0.583**	**0.533**	**0.437**	**0.379**
Pre-training over Wide Table						
Methods	PR AUC	ROC AUC	R@P=0.6	R@P=0.7	R@P=0.8	R@P=0.9
NLI	0.33	0.832	0.237	0.181	0.11	0.079
DistillBERT	0.468	0.872	0.395	0.359	0.316	0.126
Longformer	0.533	0.89	0.403	0.407	0.347	0.185
Tab-Cleaner	**0.541**	**0.903**	**0.458**	**0.411**	**0.375**	**0.3**

Table 3: Results of data cleaning over attribute applicability classification task. The numbers in bold represent the best performance. TabCleaner gives the best performance.

Pre-training over Standard Table						
Methods	PR AUC	ROC AUC	R@P=0.6	R@P=0.7	R@P=0.8	R@P=0.9
NLI	0.242	0.637	0.011	0.019	0	0
DistillBERT	0.622	**0.894**	0.561	0.388	0.226	0.011
Longformer	0.512	0.847	0.326	0.207	0.023	0.019
Tab-Cleaner	**0.623**	0.871	**0.646**	**0.476**	**0.242**	**0.059**
Pre-training over Wide Table						
Methods	PR AUC	ROC AUC	R@P=0.6	R@P=0.7	R@P=0.8	R@P=0.9
NLI	0.242	0.637	0.011	0.019	0	0
DistillBERT	0.44	0.764	0.219	0.123	0.038	0.015
Longformer	0.471	0.793	0.276	0.188	0.061	**0.019**
Tab-Cleaner	**0.487**	**0.81**	**0.415**	**0.234**	**0.076**	0.011

Table 4: Results of data cleaning over attribute value validation task. The numbers in bold represent the best performance. TabCleaner handles both short sequences and long sequences very well.

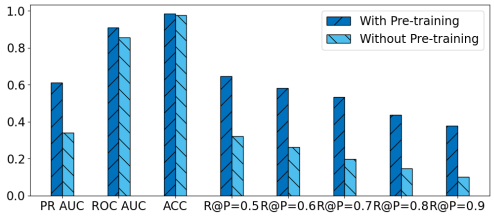

Figure 3: Improvement brought by pre-training over attribute applicability classification task.

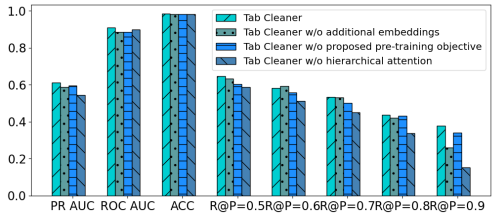

Figure 4: Impact of different components of Tab-Cleaner in terms of attribute applicability classification task.

AUC of Tab-Cleaner without pre-training from 0.340 to 0.613 and increases R@P=0.9 from 0.101 to 0.379.

Impact of different components of Tab-Cleaner Upon the base Tab-Cleaner model, we derive three different variants as follows:

- **Tab-Cleaner w/o additional embeddings:** We exclude additional embeddings during learning.
- **Tab-Cleaner w/o table structure-aware objective:** We do not employ the pre-training objective for learning tabular substructure representations.
- **Tab-Cleaner w/o hierarchical attention:** We do not adopt the hierarchical attention.

We compare these three variants with the original Tab-Cleaner framework over the attribute value validation task in Fig. 4. We observe that: (1) The original Tab-Cleaner achieves the best performance, showing the necessity of integrating all three components; (2) Tab-Cleaner w/o hierarchical attention presents the worst performance, indicating the effectiveness of the hierarchical attention mechanism in capturing information from tabular data.

4.6 Case Study

To further demonstrate the capability of Tab-Cleaner in detecting real-world errors in the Amazon dataset, we present examples of identified errors and missed errors as shown in Table 6. We pre-trained Tab-Cleaner over standard length tabular corpus and fine-tuned Tab-Cleaner over two downstream tasks: attribute applicability classification and attribute value validation. Contrary to the common settings, a positive label in an error detection scenario means the data instance is an error while a negative label means the data instance is true. Therefore, the triples with the highest probability have the highest possibility to be incorrect. Threshold σ is chosen based on the best classification accuracies on the validation dataset in order to classify the attributes. Given human labeled incorrect attributes in the test dataset, we present the top 3 attributes with the highest probability as identified errors and the top 3 attributes with the lowest probability as missed errors. We observe that attribute values of identified errors usually violate the description of products and thus can be correctly classified as errors. For example, product 1 in Table 6 is not a skin care product, thus should not have the attribute "skin type". Although the attribute values of products 7, 9, and 11 are commonly observed phrases to describe the target attributes (i.e., "dark" is widely used to describe "skin tone"), their inconsistency with the product description makes them no longer correct attribute values. We also notice that most of the missed errors are correct but labeled as errors due to the wrong annotation. We verify the correctness of all missed errors by ourselves and highlight the correct

Error Type	Identified Errors				Missed Errors			
	ID	Product	Attribute	Value	ID	Product	Attribute	Value
Inapplicable Attribute	1	Brand A Hand Sanitizer Holder Keychain	skin type	-	2	Brand B Isoprophyl Alcohol	scent	-
	3	Brand C horse Fly mask Over Fence - Face Covers	flavor	-	4	Brand D Velvetines Liquid Matte Lipstick	color	-
	5	Brand E sock stocking hose sox anklets Women Print Multicolor	age range description	-	6	Brand F Coffee, Dulce De Leche Flavored Coffee	container type	-
Incorrect Attribute Value	7	Brand G ColorStay Overtime Lipcolor Forever Scarlet (040)	skin tone	dark	8	Brand H Loose Face Powder, Translucent	finish type	matte
	9	Brand I Salad Dressing, Zesty Robusto Italian	variety	garlic	10	Brand J Advanced Defence Gum Treatment for Gingivitis	product benefit	cleansing
	11	Brand J Popcorn Seasoning, White Cheddar	Item form	butter	12	Brand K unisex-adult Bottle Bright - Hydration Pack Cleaning Tablets Clear	benefit	brightening

Table 6: Identified errors & missed errors on Amazon Data. We present the top 3 human labeled incorrect attributes with the highest probability as identified errors. Meanwhile, the top 3 human labeled incorrect attributes with the lowest probability are presented as missed errors.

attributes in red and attributes for which we cannot determine their correctness based on product profiles in blue. We observe that Tab-Cleaner can classify the samples which cannot be correctly classified by humans. This indicates the strong power of Tab-Cleaner in identifying errors.

5 Related Work

Natural Language Inference (NLI) Data cleaning for product catalog data is related to natural language inference (NLI). Given a premise (e.g., product profiles in our scenario), NLI aims to classify whether the hypothesis (e.g., attribute values in our scenario) is true, false, or undetermined (Tay et al., 2017; Chen et al., 2016). Most of the existing NLI models are based on cross-sentence attention, which can be divided into word-by-word attention-based methods (Rocktäschel et al., 2015; Wang et al., 2017; Wang and Jiang, 2015) and inter-sentence interaction-based methods (Yin et al., 2018). Existing NLI methods are typically trained on free-form natural language while Tab-Cleaner is designed to handle error detection over text-rich tabular data.

Tabular Data Representation Tables are important media of world knowledge (Cafarella et al., 2008). Motivated by the large-scale language models pretrained on tasks involving unstructured natural language, several works attempt to extend the pre-trained language models (LMs) to jointly learn representations of tables as well as text (Yin et al., 2020; Herzig et al., 2020; Zhang et al., 2019) with applications including semantic parsing (Yin et al., 2020; Herzig et al., 2020), entity linking (Deng et al., 2020) and table structure understanding (Nassar et al., 2022; Du et al., 2021; Deng et al., 2020). The training data of these works usually involve thousands of tables, where each table consists of only a few rows and columns. Our proposed method focuses on a different task, text-rich tabular data cleaning, where the training data involve only a single table w.r.t millions of rows.

Transformer for Longer Sequences It is challenging for Transformers-based models to process long sequences because their self-attention operation scales quadratically with the sequence length in terms of memory. There have been a number of attempts to alleviate this issue (Dai et al., 2019; Sukhbaatar et al., 2019; Rae et al., 2019; Wang et al., 2019; Joshi et al., 2020; Child et al., 2019), in which Longformer (Beltagy et al., 2020) and Big Bird (Zaheer et al., 2020) are the most representative methods. All these methods focus on tasks involving free-text long content (e.g., document classification, and genomics data analysis), while Tab-Cleaner is designed to cope with a wide table.

6 Conclusion

We have proposed Tab-Cleaner, a Transformer-based model designed specifically for data-cleaning tasks on text-rich tabular catalog data. It provides a versatile solution for data cleaning tasks by efficiently handling the unique challenges posed by text-rich tabular catalog data. To enhance the efficiency of training and reduce memory consumption, we have introduced a novel hierarchical attention mechanism. This mechanism enables a sparse attention pattern, allowing for the effective processing of long sequences. We train Tab-Cleaner on a real-world Amazon Product Catalog w.r.t millions of products and show that we can improve over SOTA methods greatly.

Acknowledgements

This work was partially supported by NSF 2211557, NSF 2303037, NSF 1937599, NSF

2119643, NASA, SRC, Okawa Foundation Grant, Amazon Research Awards, Amazon Fellowship, Cisco research grant, Picsart Gifts, and Snapchat Gifts.

References

Iz Beltagy, Kyle Lo, and Arman Cohan. 2019. Scibert: A pretrained language model for scientific text. arXiv preprint arXiv:1903.10676.

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. arXiv preprint arXiv:2004.05150.

Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. 2008. Webtables: exploring the power of tables on the web. Proceedings of the VLDB Endowment, 1(1):538–549.

Qian Chen, Xiaodan Zhu, Zhenhua Ling, Si Wei, Hui Jiang, and Diana Inkpen. 2016. Enhanced lstm for natural language inference. arXiv preprint arXiv:1609.06038.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. arXiv preprint arXiv:1904.10509.

Xu Chu, Ihab F Ilyas, Sanjay Krishnan, and Jiannan Wang. 2016. Data cleaning: Overview and emerging challenges. In Proceedings of the 2016 international conference on management of data, pages 2201–2206.

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators. arXiv preprint arXiv:2003.10555.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. arXiv preprint arXiv:1901.02860.

Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. Turl: Table understanding through representation learning. arXiv preprint arXiv:2006.14806.

Xin Luna Dong, Xiang He, Andrey Kan, Xian Li, Yan Liang, Jun Ma, Yifan Ethan Xu, Chenwei Zhang, Tong Zhao, Gabriel Blanco Saldana, et al. 2020. Autoknow: Self-driving knowledge collection for products of thousands of types. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 2724–2734.

Lun Du, Fei Gao, Xu Chen, Ran Jia, Junshan Wang, Jiang Zhang, Shi Han, and Dongmei Zhang. 2021. Tabularnet: A neural network architecture for understanding semantic structures of tabular data. In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, pages 322–331.

Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. Tapas: Weakly supervised table parsing via pre-training. arXiv preprint arXiv:2004.02349.

Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. 2021. Tabbie: Pretrained representations of tabular data. arXiv preprint arXiv:2105.02584.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. 2020. Spanbert: Improving pre-training by representing and predicting spans. Transactions of the Association for Computational Linguistics, 8:64–77.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Louis Martin, Benjamin Muller, Pedro Javier Ortiz Suárez, Yoann Dupont, Laurent Romary, Éric Villemonte de La Clergerie, Djamé Seddah, and Benoît Sagot. 2019. Camembert: a tasty french language model. arXiv preprint arXiv:1911.03894.

Ahmed Nassar, Nikolaos Livathinos, Maksym Lysak, and Peter Staar. 2022. Tableformer: Table structure understanding with transformers. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 4614–4623.

Jay Pujara, Eriq Augustine, and Lise Getoor. 2017. Sparsity and noise: Where knowledge graph embeddings fall short. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 1751–1756.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. 2019. Compressive transformers for long-range sequence modelling. arXiv preprint arXiv:1911.05507.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. arXiv preprint arXiv:1606.05250.

Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, and Phil Blunsom. 2015. Reasoning about entailment with neural attention. arXiv preprint arXiv:1509.06664.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108.

Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. 2019. Adaptive attention span in transformers. arXiv preprint arXiv:1905.07799.

Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. 2017. Compare, compress and propagate: Enhancing neural architectures with alignment factorization for natural language inference. arXiv preprint arXiv:1801.00102.

Shuohang Wang and Jing Jiang. 2015. Learning natural language inference with lstm. arXiv preprint arXiv:1512.08849.

Zhiguo Wang, Wael Hamza, and Radu Florian. 2017. Bilateral multi-perspective matching for natural language sentences. arXiv preprint arXiv:1702.03814.

Zhiguo Wang, Patrick Ng, Xiaofei Ma, Ramesh Nallapati, and Bing Xiang. 2019. Multi-passage bert: A globally normalized bert model for open-domain question answering. arXiv preprint arXiv:1908.08167.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. Tabert: Pretraining for joint understanding of textual and tabular data. arXiv preprint arXiv:2005.08314.

Wenpeng Yin, Hinrich Schütze, and Dan Roth. 2018. End-task oriented textual entailment via deep explorations of inter-sentence interactions. arXiv preprint arXiv:1804.08813.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. In NeurIPS.

Li Zhang, Shuo Zhang, and Krisztian Balog. 2019. Table2vec: Neural word and entity embeddings for table population and retrieval. In Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 1029–1032.

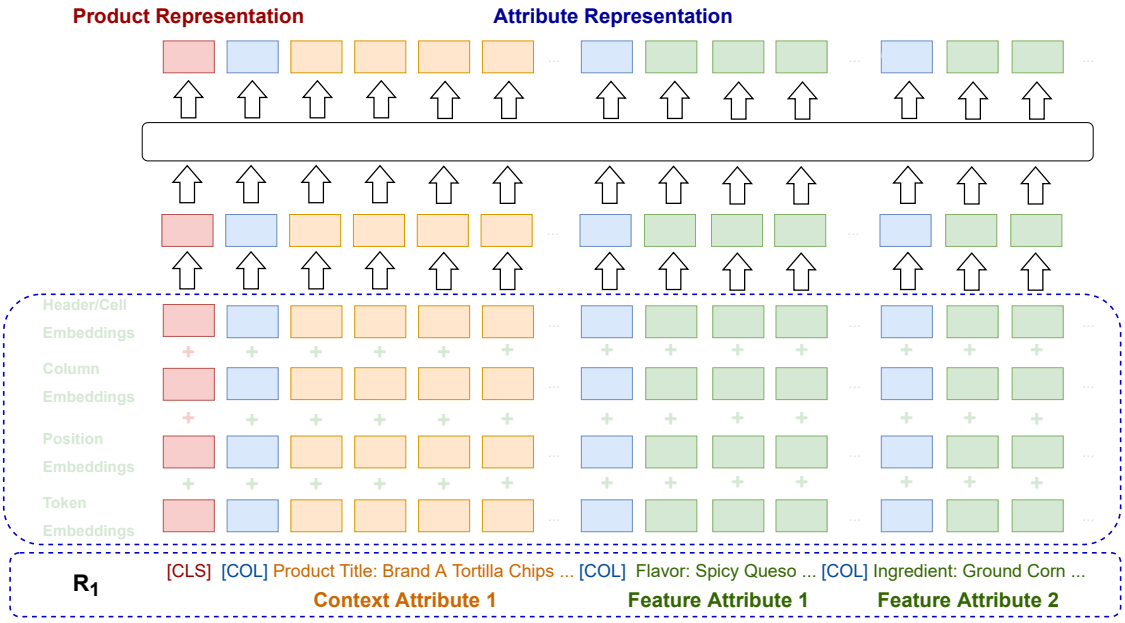

Figure 5: Example input of Tab-Cleaner. Tab-Cleaner flattens each row in the input table into a sequence of tokens. The token embeddings are combined with additional embeddings to capture tabular structure.

A Tab-Cleaner Architecture

Tab-Cleaner takes a $m \times n$ product catalog table as input and produces token representation and tabular substructure representation (i.e., cell-level representation and row-level representation). Tab-Cleaner is implemented by extending DistillBERT's architecture (Sanh et al., 2019) with additional embeddings that capture tabular structure (Fig. 5).

Additional embeddings The token embeddings are combined with additional embeddings used to encode tabular structure before feeding them to the pre-training model:

- **Position Embedding** is the relative index of a token within a cell. For example, the position embedding of k-th token in a cell is k.
- **Column Embedding** is the index of the column that the token appears in. For example, the column embedding of tokens in cell T_{ij} is j.
- **Header/Cell Embedding** indicates if the token corresponds to the column name or the attribute value. It takes two possible values: 0 for the column name and 1 for attribute values. For example, the 4-th cell of R_1 in Table 1 consists of a list of tokens {[COL], [Ingredient], [:], [Ground], [Corn], [Chipotle], [Pepper], [Powder], [Paprika], [Extract], [Spices]}, where the header/cell embedding for token [Ingredient] and [:] are 0 and 1 otherwise.

For each element x_i in the input sequence, we construct its input representation as:

$$\mathbf{h}_i^0 = \mathbf{x}_i^{\text{ele}} + \mathbf{x}_i^{\text{pos}} + \mathbf{x}_i^{\text{col}} + \mathbf{x}_i^{\text{header}} \quad (2)$$

where $\mathbf{x}_i^{\text{ele}}$ is the token embedding, $\mathbf{x}_i^{\text{pos}}$ is the position embedding, $\mathbf{x}_i^{\text{col}}$ is the column embedding, and $\mathbf{x}_i^{\text{header}}$ is the Header/Cell Embedding. After constructing all input representations, we feed them into a stack of L successive Transformer encoders to encode the sequence and obtain:

$$\mathbf{h}_i^l = \text{Transformer}(\mathbf{h}_i^{l-1}) \quad (3)$$

where $\mathbf{h}_i^l$ is the hidden state of x_i after the l-th layer.

B Implementation of the Transformer Encoder with the Hierarchical Attention

Let $\mathbf{H} = (\mathbf{h}_1, \ldots, \mathbf{h}_n)$ denote an input representation, where $\mathbf{h}_i$ is a d dimensional vector and $\mathbf{H}$ is a matrix in $\mathbb{R}^{n \times d}$. We discussed the construction of $\mathbf{H}$ in Appendix B. Given the linear projections Q, K, V, the Transformer encoder computes attention scores as follows:

$$\text{Attention}(\mathbf{H}_i) = \sum_{k=1}^{K} \sigma\big(Q_k(\mathbf{h}_i) K_k(\mathbf{H}_{\mathcal{N}(i)})^T\big).V_k(\mathbf{H}_{\mathcal{N}(i)}) \quad (4)$$

where $\mathcal{N}(i)$ denote the out-neighbors set of node i and $\mathbf{H}_{\mathcal{N}(i)}$ corresponds to the matrix over $\{\mathbf{h}_j$:

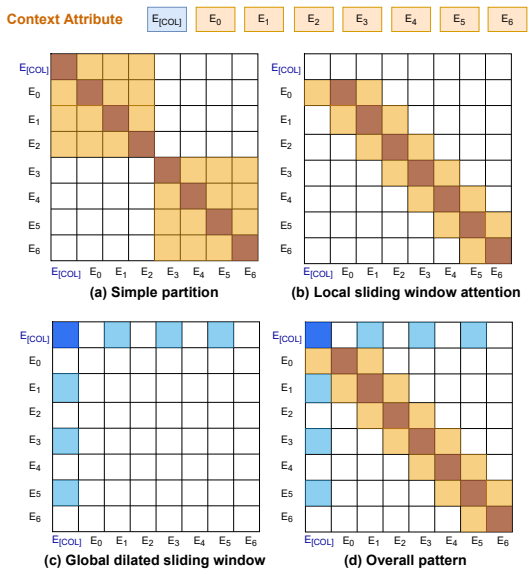

Figure 6: The configuration of attention patterns for context attribute learning. (a) partitions the long context into smaller sequences; (b) local sliding window attention ($w = 3$); (c) global dilated sliding window attention ($d = 2$); (d) the combined model.

$j \in \mathcal{N}(i)\}$. K denotes the number of heads. Q_k, K_k, V_k are query, key, and value functions. Let the adjacency matrix A define a directed graph $\mathcal{G}$. Each vertex in $\mathcal{G}$ corresponds to a token in the input sequence. $A \in [0,1]^{n \times n}$ with $A(i,j) = 1$ if query i attends to key j and is zero otherwise. The traditional Transformer encoder calculates full quadratic attention by assuming $\mathcal{G}$ is a fully connected graph. Instead, we sparsify $\mathcal{G}$ by proposing a hierarchical attention mechanism meanwhile ensure the proposed attentions are as powerful and expressive as full attention. For example, the graphical illustration of the attention pattern shown in Fig. 6 (d) is given in Fig. 7.

C Expressiveness of Local + Global Attention for Context Attribute Learning

In this section, we discussed the expressiveness of *local + global attention* and showed that it is as expressive as full attention. The graphical illustration of the attention pattern shown in Fig. 6 (d) is given in Fig. 7. Each node in the graph corresponds to a token in the input sequence. Following the proposed attention pattern, the token can only attend to its directly connected neighbors.

Definition 1 ***h-hop enclosing graph*** *For a h-hop enclosing graph $G = (V, E)$, given any two nodes $x, y \in V$, we have $d(x, y) \leq h$.*

It is obvious that the graph in Fig. 7 is a 3-hop enclosing graph. To generalize such observation, we have the following theorem.

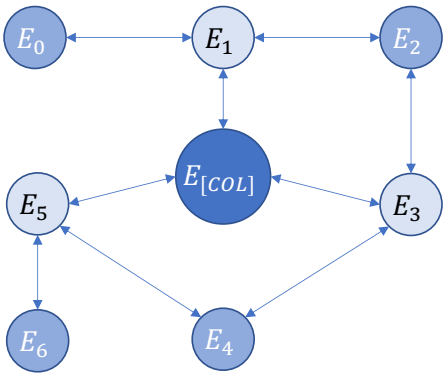

Figure 7: Graphical illustration of attention pattern shown in Fig. 6 (d). Each node in the graph corresponds to a token in the input sequence. The token can only attend to its directly connected neighbors.

Theorem 1 *Given an input sequence with length as L, assuming we set dilation equal to the window size $d = w = ceil(\sqrt{L})$, its attention graph is always a 4-hop enclosing graph.*

We denote the input sequence as $x_{0:L-1} = (x_0, \ldots, x_{L-1})$. Given a fixed window size w, each token attends to $1/2w$ its local neighboring tokens on each side and the global token [COL] (x_0) attends tokens $(x_0, x_d, \ldots, x_{n*d})$ where $n = \text{ceil}(\sqrt{L}) - 1$. Next, we will show that given any node $x_i \in V$, we have $d(x_i, x_0) \leq 2$.

- If $i = k * d$, token x_i directly connected to the global token [COL]. We have $d(x_i, x_0) = 1$.
- If $i \neq k * d$, token x_i attends all tokens within the window $x_{(i-1/2*d):(i+1/2*d)}$ (we set $w = d$). We can always find an integer k which satisfies $(i - 1/2 * d) \leq k * d \leq (i + 1/2 * d)$ because $(i + 1/2 * d) - (i - 1/2 * d) = d$. Therefore, we have $d(x_i, x_0) = 2$.

Since $d(x_i, x_0) \leq 2$, we have $d(x_i, x_j) \leq d(x_i, x_0) + d(x_j, x_0) = 4$. The attention graph is always a 4-hop enclosing graph.

We know that a node in an h-hop enclosing graph is able to collect information from any other node in the graph using an h-layer GNN. Therefore, any token x_i is able to aggregate information from all tokens in the sequence using a GNN with over 4 layers.

D Conditional Encodings of Feature Attributes over Context Attributes

Note that content attributes contain rich information about products, which is useful for verifying the correctness of feature attributes. For example, the product title *"Brand A Tortilla Chips Spicy Queso, 6 - 2 oz bags"* covers multiple attributes, including brand, product category, flavor, and size. We can easily verify the correctness of these attributes against the product title. However, given the hierarchical attention mechanism, feature attributes are learned independently from context attributes during the cell encoding stage. Although the correlations between feature attributes and context attributes can be captured afterward during the row encoding stage, the cross-cell token-to-token correlation is lost. To enable cross-cell token-to-token conditional encoding of feature attributes over context attributes, we improve cell encoding for feature attributes as shown in the following example.

Example: Given a product with description *" Mango Chipotle Origami Wraps are all natural sushi wraps made from vegetable and fruit purees. This wrap has* ***a ripe, tropical mango flavor balanced with the bold spiciness of chipotle pepper****. Origami Wraps are healthy, vegan, gluten-free alternatives to seaweed nori and/or soy paper. They are a creative, flavorful, and colorful new ingredient for restaurant and home chefs alike. Use Mango Chipotle wraps to add some* ***Latin fusion flavor*** *to traditional sushi or to create innovative sushi-style rolls with many different non-seafood ingredients. Can be used for onigiri, nigiri, and musubi."*, only the words highlighted in **boldface** describe target attribute *flavor*. The value of this product on attribute *flavor* is *"mango"*, which is given in the product catalog.

To capture the cross-cell token-to-token correlation between feature attributes and context attributes, the most straightforward way is to concatenate the context attributes and target feature attribute and require each token in the target feature attribute to attend the entire concatenation. Since the concatenation is usually long, attending all tokens in the concatenation is computationally impractical. Note that most contents in context attributes are irrelevant to the target feature attributes. We extract only the relevant information from context attributes to build the concatenation instead. We adopt two different extraction strategies as follows:

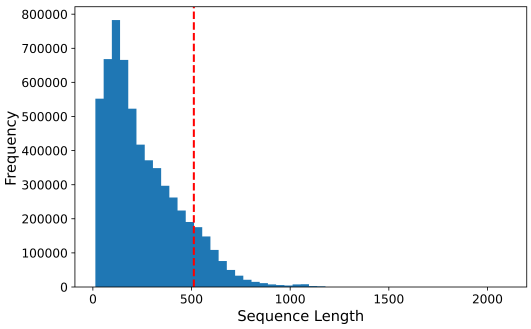

Figure 8: Length distribution of string encoding for over 3 million randomly sampled products.

- Extract the words around the target attribute value. Given the above example, the words highlighted in blue will be extracted to concatenate with *"Flavor: Mango"*.
- Extract the words around the most frequently observed textual values for the target attribute. Assuming the most frequent observed textual values for attribute *flavor* include { *"fruit"*, *"spiciness"*, *"chocolate"*, *"vanilla"* }, the words highlighted in red will be extracted to concatenate with the *"Flavor: Mango"*.

E Experiments

E.1 Analysis of Amazon Data

Tab-Cleaner flattens each row in the input table into a sequence of tokens by concatenating all textual attributes. Such a process may raise a super long sequence. To investigate the length of rows in commonly used catalogs in daily business, we randomly sampled web pages from the Amazon website and extract dozens of commonly observed attributes to construct a standard table. We show that long sequences have been widely observed in the standard table with only dozens of attributes. To better understand the performance of Tab-Cleaner over extremely long sequences, we further construct a wide table, which contains hundreds of attributes.

Analysis To investigate the length of rows in Amazon data used in daily business, we randomly sampled over millions of products associated with 31 commonly used attributes. To avoid bias, we follow the product categories' frequency distribution (i.e., commonly-occurring product categories are sampled more often than rare product categories) to sample data. The sampled products are cross hundreds of product categories from different domains, such as food, beauty, and drug. After concatenat-

Dataset	Standard Tabular Data	Longer Sequence
#Attributes (Columns)	31	119
#Products (Rows)	3,110,715	677,744
#Context Attributes	7	7
#Feature Attributes	24	112
#Average length	257	639

Table 7: Pre-training data statistics.

ing all attributes of a product into a sequence, the length distribution of products' string encoding is given in Fig. 8. As observed from the figure, even though we include only 31 attributes, over 10% rows have lengths over 512. Note that most existing Transformer models can only handle sequences that fall within the typical 512-token limits, it is necessary to scale Tab-Cleaner to process longer input sequences.

E.2 Datasets

Datasets for Pre-training To prepare the unlabeled product catalog corpus for pre-training, we construct two tables based on randomly sampled web pages from the Amazon website. The first table contains a standard amount of attributes (i.e., 31). We call it a standard table. The second table contains a much larger amount of attributes (i.e., 119). We call it a wide table. The detailed statistics about these two tables are given Table 7.

Standard Table We construct the standard table using the data sampled in Section E.1. After flattening the table into a sequence of tokens, the average length of rows in the standard table is 257.

Wide Table To better investigate the performance of Tab-Cleaner over extremely long sequences, we construct a wide table that contains hundreds of attributes. To ensure the sufficient length of sampled data, we concatenate all attributes of a product into a sequence and select only products with sequence lengths over 512. The wide table contains 677,744 products associated with 119 attributes. The average length of rows in the wide table is 639.

Datasets for Fine-tuning To prepare labeled data for fine-tuning, we asked Amazon Mechanical Turk (MTurk) workers to manually label the correctness of attributes based on product profiles. Each data point is annotated by three Amazon Mechanical Turk workers and the final label is decided by majority voting. In order to ascertain the performance of learned Tab-Cleaner representation over error detection, we study two downstream tasks: attribute applicability classification and attribute value validation. As shown in Table 8, the labeled

Task	# Feature Attributes	Data Split		
		#Train	#Validation	#Test
Attribute Applicability Classification	24	43,002	2,150	8,601
Attribute Value Validation	26	7,770	309	1,235

Table 8: Fine-tuning data statistics

data for the attribute applicability classification task covers 53,753 products and 24 feature attributes, and the labeled data for the attribute value validation task covers 9,713 products and 26 feature attributes. For both datasets, 80 percent of the data is used as training data for fine-tuning and the rest is used as validation and test data. Contrary to the common settings, a positive label in an error detection scenario means the data instance is an error while a negative label means the data instance is true. We can observe that both datasets are super unbalanced. Only a few data are labeled as errors (i.e., positive labels).

E.3 Experimental Setup

Pre-training & Fine-tuning Before pre-training, we truncate each row's content to satisfy the maximum sequence length requirement, as some rows contain huge amounts of text. We train Tab-Cleaner for three epochs for pre-training. Tab-Cleaner has 6 layers, a hidden dimensionality of 768, and 12 heads. We use the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 5e-5. For fine-tuning, we initialize the parameters with the pre-trained model, and further train all parameters with a binary classification objective for 10 epochs. To fairly compare different transformer-based methods, the same setting is employed for all models. We build data for evaluation using the held-out validation/test rows to ensure that there is no overlapping data in training and validation/test.

Evaluation Metric. We adopt the area under the Precision-Recall curve (PR AUC), area under the Receiver Operating Characteristic Curve (ROC AUC), and Recall at Precision=X (R@P=X) to evaluate the performance of the models over error detection. To be more specific, PR AUC is defined as the area under the precision-recall curve, which is widely used to evaluate the ranked retrieval results. ROC AUC is a performance measurement for classification problems at various threshold settings, telling how much the model is capable of distinguishing between classes. R@P is defined as the recall value at a given precision, which aims to evaluate the model performance when a specific precision requirement needs to be satisfied. For

example, R@R = 0.7 shows the recall when the precision is 0.7.

Compared Methods. We evaluate Tab-Cleaner against state-of-the-art (SOTA) algorithms, including (1) DistillBERT (Sanh et al., 2019) since Tab-Cleaner is implemented by extending DistillBERT; (2) Transformer for Longer Sequences (e.g., Longformer (Beltagy et al., 2020)); (3) nature language inference (NLI) methods. The SOTA Transformer for NLI is selected as our baseline. In our setting, the input of the NLI model includes two parts: product profiles (i.e, concatenation of context attributes) and the corresponding feature attribute values. These two sequences are concatenated using a separator token ([SEP]). The first token of input is always set as a special token ([CLS]). We feed the input into Transformers. The final hidden state corresponding to [CLS] is used as the final representation. To predict the correctness of the attribute, a binary classifier is placed over the [CLS] representation for inference. All baseline methods are built within HuggingFace's framework.

We did not include tabular representation learning models as our baseline because existing tabular representation learning models focus on different downstream tasks such as table query (i.e., answering either SQL questions or natural language questions given a table) or tabular structure prediction (i.e., predict the data type or tag of a cell). They cannot be applied to clean catalog data. First, they require a different input data format. For example, table query requires paired tables and text (e.g., natural language questions and their answers) and tabular structure prediction requires the tags of cells. Second, they can only deal with tiny tables (e.g., TABBIE (Iida et al., 2021) has to truncate tables to 30 rows and 20 columns).

E.4 Data Cleaning Tasks

Attribute Applicability Classification To evaluate whether the proposed hierarchical attention mechanism is as powerful and expressive as full-attentions, we pre-train each method over a standard-length tabular corpus (standard table), where most of the rows satisfy the maximum sequence length requirement (512) without truncation. A binary classifier is trained to predict the correctness of attributes during fine-tuning. We also pre-train Tab-Cleaner over a longer tabular corpus (wide table), which has contexts significantly longer than 512 tokens.