NOVA IMS Information Management School

MDSAA

Master's Degree Program in

Data Science and Advanced Analytics

Large Language Model for Semantic Table Interpretation

LongSemAnnotator: A Longformer-based Framework Using Inter-Table Context for Column Type Annotation

Tiago Moço dos Santos

Master Thesis

presented as partial requirement for obtaining a Master's Degree in Data Science and Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School

Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

Large Language Model for Semantic Table Interpretation

LongSemAnnotator: A Longformer-based Framework Using Inter-Table Context for Column Type Annotation

by

Tiago Moço dos Santos

Master Thesis presented as partial requirement for obtaining the Master's degree in Data Science and Advanced Analytics, with a specialization in Data Science.

Supervisor: Mauro Castelli

Co-Supervisors: Fernando Augusto Junqueira Peres, Yuriy Perezhohin

July, 2024

Statement of Integrity

I declare that I have carried out this academic work with integrity. I confirm that I have not resorted to plagiarism or any other form of misuse of information or falsification of results during the process of preparing this work. I also declare that I am aware of the Rules of Conduct and the Code of Honor of the NOVA Information Management School.

Tiago Santos

Lisbon, 15 of July of 2024

Acknowledgements

First and foremost, I am deeply grateful to my parents, whose constant support throughout my life has made this achievement possible. Their encouragement and guidance have been fundamental in my academic journey.

I extend my sincere gratitude to my supervisors, Mauro Castelli, Fernando Peres, and Yuriy Perezhohin, for their expert guidance and mentorship throughout this complex process. Their support and the opportunities they provided have been priceless to my growth, both personally and professionally. A special thanks to "Door Whisper" Yuriy for his insightful advice, belief in my potential, and the many Portela coffees.

I am also thankful for the support of my friends, who have shared in my moments of both frustration and joy throughout this process. Particularly, thanks to Beatriz, even though you drive me crazy most of the time, you were a core piece in this and helped me keep my balance.

I would also like to express my gratitude to Victor Costa and the Analytics Lab of NOVA IMS for their support and provision of the necessary infrastructure that enabled me to successfully develop my experiments.

Finally, I would like to acknowledge the dedicated faculty at NOVA University, whose commitment to teaching and mentorship has enriched my academic experience and contributed to the completion of this thesis.

Abstract

Column-Type Annotation (CTA) is a crucial task in Semantic Table Interpretation (STI), with applications in data integration, search, and Knowledge Graph (KG) construction. Existing methods often struggle to handle large tables or incorporate contextual information from multiple tables. This thesis introduces a novel framework for STI that addresses these limitations.

Our framework utilizes sentence transformers to generate semantic embeddings for columns, enabling the identification of semantically similar columns across tables. We then employ a Longformer-based model, capable of processing long input sequences, to incorporate this inter-table context into the annotation process.

We evaluate our framework on the SOTAB benchmark, a dataset designed for CTA. The results demonstrate that our approach outperforms or matches state-of-the-art models in most test scenarios, particularly when trained on smaller datasets. We further analyze the model's performance on various test sets, including those with missing values, format heterogeneity, and corner cases.

Our findings reveal that the inclusion of inter-table context and the use of the Longformer model significantly improve CTA accuracy. We also identify challenges in annotating certain column types with high semantic overlap, highlighting potential areas for future research. Overall, this work contributes to the advancement of STI by providing a more accurate, robust, and scalable framework for column type annotation.

Keywords

Semantic Table Interpretation, Large Language Models, Semantic Similarity, Natural Language Processing, Tabular Data, Transfer Learning

Table of Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence

ANNs Artificial Neural Networks

BERT Bidirectional Encoder Representations from Transformers

CTA Column-Type Annotation

DL Deep Learning

ELMo Embeddings from Language Models

GPT Generative Pre-Training Transformer

IE Information Extraction

KB Knowledge Base

KG Knowledge Graph

LSTMs Long Short-Term Memory Networks

MLM Masked Language Modeling

MLP Multi-layer Perceptron

ML Machine Learning

NLG Natural Language Generation

NLP Natural Language Processing

NLU Natural Language Understanding

NN Neural Networks

PTMs Pre-Trained Models

RNNs Recurrent Neural Networks

RoBERTa Robustly Optimized BERT Pretraining Approach

Seq2Seq Sequence-to-Sequence

SOTAB Schema.org Table Annotation Benchmark

STI Semantic Table Interpretation

T5 Text-to-Text Transfer Transformer

TaLMs Tabular Language Models

TL Transfer Learning

Chapter 1

Introduction

The vast amount of knowledge embedded in relational tables spread across the World Wide Web holds immense potential for a variety of applications, from enhancing search engine results to powering intelligent question-answering systems. STI, the field dedicated to extracting this knowledge, has evolved from traditional, hand-crafted methods to powerful Deep Learning (DL) models. However, current state-of-the-art models still face significant challenges, particularly in handling large datasets and incorporating contextual information across multiple tables.

The evolution of STI from traditional, heavily engineered task-specific methods, such as simple statistical approaches Zhang and Balog (2017), to contemporary DL-based models Hulsebos et al. (2019) has expanded the possibilities in this field. The application of STI holds tremendous potential, setting the stage for the development of innovative semantic-based services. These services encompass a spectrum from refining dataset indexing in search engines, augmenting question-answering systems, and streamlining dataset recommendations.

Relational tables serve as repositories of substantial knowledge, organizing entities and attributes. This trend is evident in the aggregation of extensive collections of such tables, showcasing a growing acknowledgment of the valuable insights embedded within organized datasets. For instance, Bhagavatula et al. (2015) extracted 1.6 million high-quality relational tables from Wikipedia, highlighting ongoing efforts to curate and utilize these datasets. The huge amounts of data on these datasets and powerful computational resources have played a crucial role in advancing DL approaches.

Over the past few decades, those approaches become more attractive in the STI community, and is possible to identify two main directions for the application of DL in the STI domain:

- **Knowledge Graph Modelling:** Adopts an entity-level strategy, focusing on learning embeddings for entities within table cells using Knowledge Graphs embedding techniques like TransE (Bordes et al. (2013)) and TransH (Wang et al. (2014)). It encodes entities and relationships into a vector space, with a core belief that entities in the same column should exhibit semantic similarities.

- **Table Modelling:** Takes an approach by considering the textual content of tables, intra-table, and inter-table interactions. It leverages deep neural networks or language models like Bidirectional Encoder Representations from Transformers (BERT) (Suhara et al. (2022), Deng et al. (2022), Hulsebos et al. (2019)) to

learn contextualized representations for fundamental table elements such as cells and columns. This broader perspective allows for a more nuanced understanding of the relationships within and between tables.

However, the widespread adoption of BERT-like models such as TURL (Deng et al. (2022)) and DoDuo (Suhara et al. (2022)) face a significant challenge: the 512-token limit. This constraint, primarily due to the quadratic complexity of the self-attention mechanism in the vanilla Transformer architecture, necessitates substantial computational resources for longer input sequences. Consequently, many existing models treat tables in isolation, neglecting the potential contextual relationships between tables within a database or web page. This oversight hinders a holistic understanding of STI.

In this thesis, we specifically tackle the Column-Type Annotation (CTA) task within Table Modeling, addressing the challenges associated with textual content, inter-table interactions, and wide tables. Our approach leverages the power of the Longformer model (Beltagy et al. (2020)), a transformer architecture designed to handle longer input sequences, thus mitigating the limitations of BERT-like models. Furthermore, we incorporate inter-table context, drawing insights from advancements like RECA (Sun et al. (2023)) and TCN (Wang et al. (2021a)), to improve annotation quality by reducing ambiguity and capturing nuanced relationships between tables.

1.1 Objectives

The main objective of this master's thesis is to advance STI by introducing LongSemAnnotator, a novel Longformer-based (Beltagy et al. (2020)) framework for the CTA task. We hypothesize that this framework, integrating inter-table context and semantic similarity, will outperform existing BERT-based approaches on large datasets containing wide tables. All artifacts and code for LongSemAnnotator are available on GitHub[1] for reproducibility and further exploration within the STI community.

This research seeks to improve the field of STI by comparing the efficacy of transformer architectures, specifically Longformer's innovative attention mechanisms, with traditional STI approaches using BERT. Longformer's ability to handle global attention will be explored for its potential benefits in processing wide tables and managing vast amounts of data. This comparative investigation will contribute valuable insights into the strengths and limitations of attention mechanisms in STI, establishing the foundation for the development of more effective models and methodologies. This is achieved by providing comprehensive answers to the following research questions:

- **RQ1 -** How can semantic similarity be effectively used to identify related columns and incorporate inter-table context in column type annotation CTA tasks?
- **RQ2 -** What is the impact of varying input token lengths and model configurations on the performance of Longformer-based CTA models?
- **RQ3 -** How does transfer learning, specifically fine-tuning a pre-trained language model, impact the performance of CTA models, especially when dealing with limited labeled data?

[1] GitHub Repository.

- **RQ4 -** What are the challenges and limitations of current CTA models, particularly in handling specific column types and scenarios with limited context or data?
- **RQ5 -** How does our proposed Longformer-based CTA framework, which incorporates inter-table context and semantic similarity, compare to existing state-of-the-art models on a standard benchmark dataset?

1.2 Document Structure

This thesis is organized to provide a detailed exploration of the proposed CTA framework. Each chapter builds upon the previous one, culminating in a thorough evaluation and discussion of the framework's performance, implications, and potential future directions.

- **Chapter 2:** This chapter establishes the foundational knowledge necessary to understand the concepts and techniques employed throughout the thesis. It provides definitions of key terms related to STI and the proposed framework, ensuring clarity and consistency in subsequent discussions.
- **Chapter 3:** This chapter presents a full review of relevant literature, including existing word embedding techniques and models used in tabular data-related tasks. It situates our work within the broader research field, highlighting the gaps and limitations that our framework aims to address.
- **Chapter 4:** This chapter details the methodology used to develop and evaluate our Longformer-based CTA framework. It covers the selection of pre-trained models, data preprocessing techniques, model architecture, training procedure, and evaluation metrics.
- **Chapter 5:** This chapter elaborates on the core algorithms of our framework, encompassing the training and inference procedures for column type annotation CTA. It details how the model learns relationships between columns during training and utilizes this knowledge for accurate annotation of new, unseen tables.
- **Chapter 6:** This chapter presents the empirical results of our experiments, including the performance of the model under different configurations and on various test sets. We analyze these results in depth, exploring the impact of different factors on the model's accuracy and robustness.
- **Chapter 7:** In this final chapter, we summarize the key findings of our research, reiterate the contributions of the thesis, and discuss the implications of our work for the field of STI. We also outline potential research trajectories, suggesting ways to build upon and extend our framework to address remaining challenges and broaden its applicability.

Chapter 2

Background

2.1 Tabular Data

This thesis primarily focuses on structured or tabular data, which is represented as tables consisting of rows and columns. In this format, each row represents an individual record or sample, and each column corresponds to a specific attribute. The cell, defined as the intersection of a row and a column, serves as the fundamental unit of a table. Tabular data is widely used in relational databases for efficient data organization and on web pages to present information clearly to users. According to Singh and Bedathur (2023), tabular data possesses distinct characteristics that differentiate it from other data formats, key features include:

- The cell as the fundamental unit, holding discrete values.
- Consistency in data types within each column.
- Rotational invariance, meaning that the order of columns does not affect the table's interpretation.
- Uniqueness of each row, where repeated rows hold no special significance.

In the present era, we are witnessing an unprecedented surge in data generation across diverse domains. This wealth of information manifests in various forms, and we can categorize data into three primary formats:

- **Structured Data** - This form of data is meticulously organized, typically presented in tables with rows and columns. Each column represents data of the same attribute, ensuring a systematic structure.
- **Unstructured Data** - In contrast, unstructured data is often text-heavy and lacks a predefined structure. Analyzing, processing, and extracting insights from unstructured data can be challenging due to its inherent complexity.
- **Semi-Structured Data** - This type of data reaches the ideal balance between structure and flexibility. It often includes markers or tags, providing a flexible hierarchy of records and fields.

2.2 Machine Learning

Machine Learning (ML), a subfield of Artificial Intelligence (AI), is dedicated to constructing systems that learn from data. Unlike traditional rule-based systems, ML models autonomously analyze vast datasets and make decisions or predictions based on patterns they learn from the data. ML enables computers to recognize complex patterns and solve tasks, often surpassing human performance in terms of scale and processing speed. We can categorize ML into several learning paradigms:

- **Supervised Learning** - This paradigm involves training models using labeled data, where the model learns to map inputs to corresponding outputs. Examples include Decision Trees Quinlan (1986) and Support Vector Machines Hearst et al. (1998).
- **Unsupervised Learning** - Algorithms in this category operate on unlabeled data to discover inherent patterns and structures. Techniques like K-Nearest Neighbors Cover and Hart (1967) and Self-Organizing Maps Vesanto (1999) are used for pattern discovery and clustering.
- **Reinforcement Learning** - This paradigm focuses on training agents to make sequential decisions by interacting with an environment, guided by rewards and penalties. Q-Learning Watkins and Dayan (1992) is an example of a reinforcement learning algorithm.

2.3 Deep Learning

DL, a subset of ML, emulates the structure and functions of the human brain through Artificial Neural Networks (ANNs). This approach has led to exceptional performance across various fields, such as Natural Language Processing (NLP), computer vision, and image processing. A critical feature of DL is its ability to extract salient features from raw data for specific tasks autonomously. This capability significantly differs from traditional ML models, which primarily depend on manually intensive feature engineering. The journey of Neural Networks (NN) began with the perceptron model, conceptualized by Rosenblatt (1958). While effective for binary classification tasks, perceptrons had limitations when it came to handling non-linear problems. This led to the development of the Multi-layer Perceptron (MLP), which introduced multiple hidden layers and non-linear activation functions. An MLP is a structured network with an input layer, an output layer, and numerous hidden layers, where each neuron processes inputs via activation functions. Backpropagation, an iterative method, is essential in NN learning, updates the network's weights by calculating the gradient of the loss function, which assesses the discrepancy between the network's output and the actual label. This journey towards complexity and efficiency in NN paved the way to more complex architectures such as Recurrent Neural Networks (RNNs) Rumelhart et al. (1988).

2.4 Natural Language Processing

NLP stands as a branch of artificial intelligence, dedicated to transforming unstructured text data into organized representations across various linguistic levels. This transfor-

mation equips computers with the capability to understand and interpret human-like language, opening up a multitude of applications across diverse domains. As elucidated by Liddy (2001), navigating linguistic structures involves traversing seven fundamental levels, each escalating in complexity, as detailed in Figure 2.1. NLP encompasses two primary subfields, each addressing distinct aspects of language comprehension and generation: Natural Language Understanding (NLU) and Natural Language Generation (NLG).

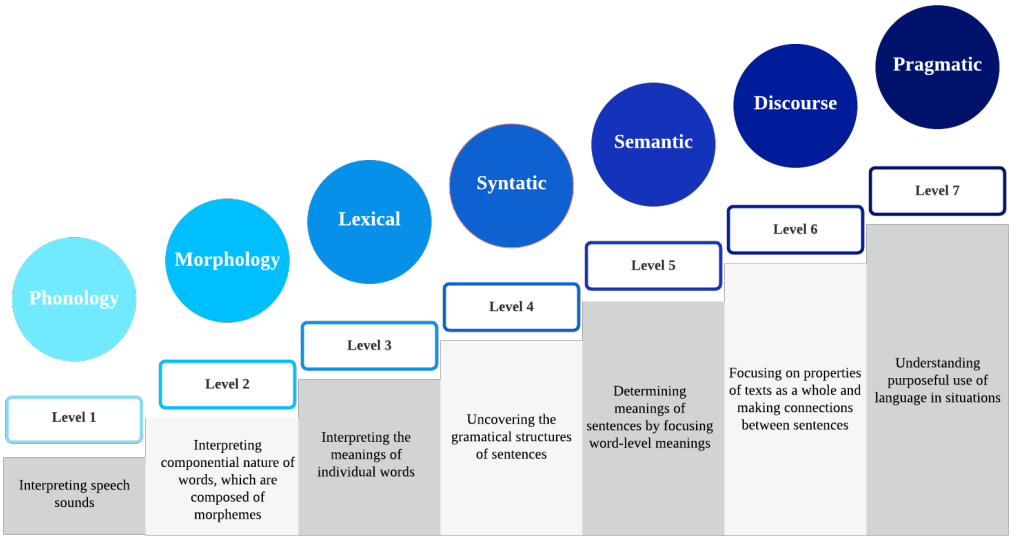


Figure 2.1: Seven levels of linguistic structure.

2.4.1 Natural Language Understanding

NLU seeks to empower machines the ability to comprehend and interpret human language, similar to human cognition. This involves extracting meaning and intent from unstructured text data, enabling computers to discern context, sentiment, and linguistic nuances. Techniques within NLU encompass semantic analysis, syntactic parsing, and named entity recognition. However, NLU faces the inherent challenge of ambiguity in human language. Ambiguity, according to Sennet (2023) can manifest in various forms, but we are just highlighting the most common:

- **Lexical Ambiguity or Polysemy** - Occurs when a word has multiple meanings, such as ”bat” being both a noun and a verb.
- **Syntactic Ambiguity** - Arises from the structure or arrangement of words in a sentence, leading to multiple valid interpretations based on grammar. For instance, ”I

saw the man with the telescope." can be interpreted as the speaker using a telescope to see the man, or seeing that the man had a telescope.

- **Semantic Ambiguity** - Persists even after resolving syntax and meanings, offering two ways of reading a sentence based on contextual understanding. For example, "John and Mary are married.", are they married to each other or are they married in separate marriages?

2.4.2 Natural Language Generation

Conversely, NLG focuses on generating coherent and contextually relevant human-like language text that is not only grammatically correct but also contextually fitting and meaningful. Techniques within NLG encompass text planning, sentence structuring, and lexical choice for creating narratives, summaries, or responses. NLG faces its own challenges, including data quality and bias concerns, evaluation metrics, and ethical considerations. Models trained on large-scale datasets can unintentionally perpetuate existing biases, which might result in the generation of biased or inappropriate content. Additionally, the development of effective methods to assess the quality of generated language remains a significant challenge. Traditional metrics, like BLEU Papineni et al. (2002), often fall short in capturing the subtleties and nuances inherent in natural language.

2.5 Sequence Models

Sequence models have emerged as a cornerstone in the field of NN, particularly in the context of NLP. This section provides a complete overview of these models, tracing their historical development, core concepts, challenges, and advancements.

2.5.1 Recurrent Neural Networks

RNNs, represent a foundational development in processing sequential data. Their unique architecture, characterized by self-loops, enables them to retain information from previous inputs, making them particularly effective for context-dependent tasks, such as text processing. However, RNNs has limitations, particularly the challenges of exploding and vanishing gradients. The issue of exploding gradients arises when gradients grow excessively, causing instability during training. On the other hand, vanishing gradients occur when gradients become exceedingly small, impeding effective information transfer within neural networks. These limitations become more prominent when handling sequences of extensive length, a common scenario in language modeling.

2.5.2 Long Short-Term Memory Networks

The introduction of Long Short-Term Memory Networks (LSTMs) in Hochreiter and Schmidhuber (1997) marked a significant advancement. LSTMs address the limitations of RNNs through a sophisticated gating mechanism, comprising input, output, and forget gates, that regulate information flow. This architecture enhances the model's capacity to

preserve essential long-term information while discarding the irrelevant, effectively overcoming the challenge of learning dependencies. LSTMs have played a crucial role in language modeling, enabling models to maintain context across extended content.

2.5.3 Sequence-to-Sequence Models

Using a single LSTM layer poses challenges in handling task complexity, capturing long-term dependencies, and achieving a limited representational capacity. These challenges are effectively addressed by adopting a multi-LSTM approach, as seen in Sequence-to-Sequence (Seq2Seq) models. Introduced in Sutskever et al. (2014), Seq2Seq models consist of two components: an encoder and a decoder, typically built with LSTMs units. The encoder condenses the input sequence into a context vector, and the decoder generates the corresponding output sequence. As seen in machine translation tasks, this structure transforms one sequence into another. However, the fixed-size context vector produced by the encoder can be a bottleneck when processing extensive sequences. Attention mechanisms were introduced in Bahdanau et al. (2014) to address this issue, allowing the decoder to access the entire input sequence at each step, significantly enhancing the model's performance.

2.5.4 Transformers

The introduction of the Transformer architecture by Vaswani et al. (2023) represents a significant departure from traditional recurrence-based models. Its innovative self-attention mechanism allows simultaneous focus on different positions in input sequences, enabling parallelization and reducing training time. Unlike Seq2Seq models, Transformers handle long-range dependencies effectively, enhancing contextual understanding. Transformers offer customization through pre-training and fine-tuning, adapting well to tasks with limited domain-specific data. Their versatility supports multi-task learning, improving efficiency and promoting a detailed understanding of diverse input data. The speed advantage of Transformers, attributed to parallelization capabilities, makes them powerful tools for natural language processing and sequence-based applications.

2.6 Vanilla Transformer

As discussed earlier, the introduction of the Transformer architecture brought about a significant departure from sequential processing, characteristic of traditional Seq2Seq models. While Seq2Seq models process inputs sequentially, leading to longer training times and challenges in handling long-range dependencies, the Transformer adopts a parallel processing approach. This method allows for the simultaneous processing of training data, aligning well with the architecture of GPUs and considerably accelerating the training process.

2.6.1 Fundamentals of Transformer Architecture

The Transformer's architectural synthesis enables the model to systematically capture hierarchical and complex patterns within the input data, contributing to its ability to under-

stand and process complex information, setting new benchmarks in numerous NLP tasks. Understanding the foundational elements is essential as the basis for constructing both the encoder and decoder within the Transformer Architecture. This architectural synthesis is represented in Figure 2.3, where the encoder occupies the left half and the decoder the right.

Positional Encoding

Positional encoding becomes crucial when considering the parallel processing of input data in Transformers. Without positional encoding, the model might lose sense of the order of input data and treat identical words in different positions as equivalent. This encoding enables the model to distinguish between occurrences based on their positions, even when their semantic meanings are identical. For example, it helps distinguish between sentences like "Tiago eats pizza" and "Pizza eats Tiago," preventing potential confusion by generating unique vectors for each instance based on their positions. The original Transformer paper introduced positional encodings that utilize periodic sine and cosine functions, considering factors such as word position pos, embedding dimension i, and model dimension d_{model}, the functions are calculated as:

$$PE_{(\text{pos},2\text{i})} = \sin\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right) \tag{2.1}$$

$$PE_{(\text{pos},2\text{i+1})} = \cos\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right) \tag{2.2}$$

Self-Attention and Multi-Head Self-Attention Mechanism

The core of the Transformer's effectiveness lies in its self-attention mechanism. This mechanism processes each input word using three distinct vectors: Query, Key, and Value. It computes the dot product between the Query vector and Key vectors of all other embeddings, normalized for dimensional alignment. The softmax function creates a probability distribution indicating the relative importance of each word embedding for the current word. This output transforms each Value vector, yielding new, contextually enriched vectors, the process To capture diverse relational aspects among words in a sequence, Transformer employ multi-head self-attention layers. Each head, with its Query, Key, and Value matrices, learns different dimensions of inter-word relationships. The resulting matrices from all heads are concatenated and multiplied by a jointly trained weight matrix, synthesizing a representation of the input sequence.

Residual Connection, Normalization Layer, and Feedforward Layer

The Transformer block, centered around the self-attention layer, incorporates additional elements for enhanced performance. Residual connections facilitate information flow between lower and higher layers, mitigating the vanishing gradient problem and aiding training convergence. Normalization layers stabilize the training process, making it robust to variations in input distributions. Feedforward layers process each position independently, capturing complex patterns and relationships within the input data.

Masked Multi-Head Self-Attention

In the decoder component of the Transformer architecture, a masked self-attention mechanism is introduced. This mechanism restricts the model from attending to future positions in the sequence, ensuring that each position in the sequence is attended to based solely on information available up to that point. This aligns with the autoregressive property during training, making it suitable for tasks like language generation. To illustrate this mechanism, as well as multi-head self-attention in both the encoder and decoder, let's analyze Figure 2.2. These visual representations were generated using the BertViz tool, showcasing its application in a machine translation task where a sentence is translated from English to French.

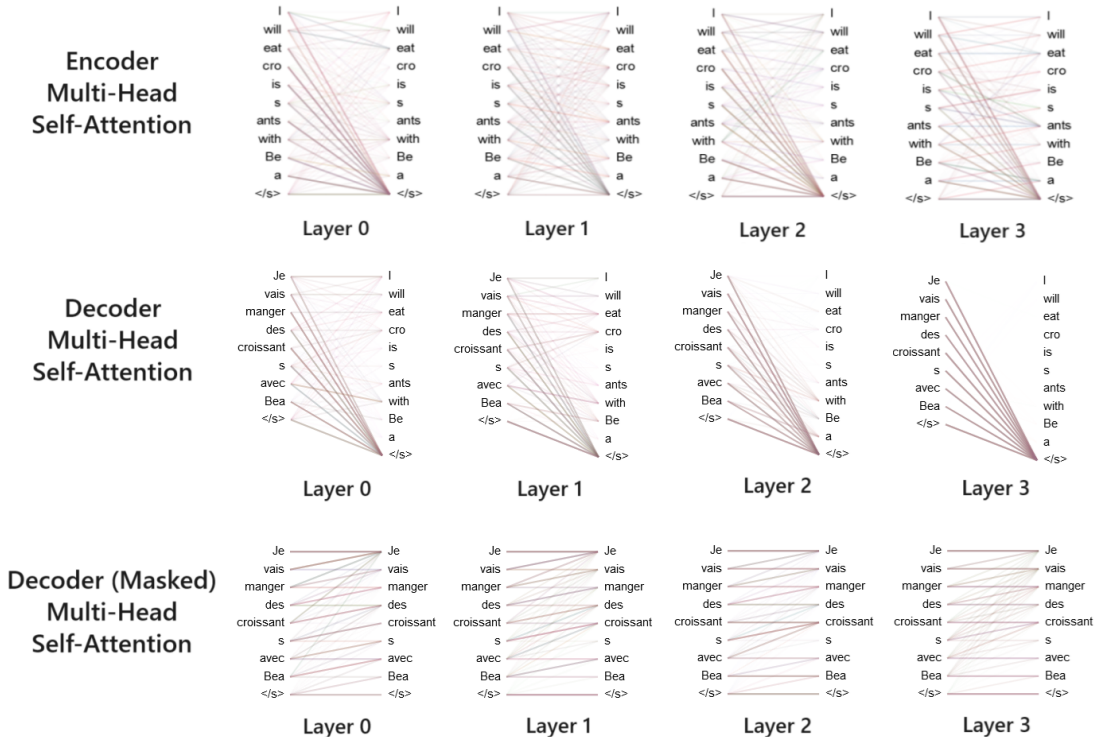


Figure 2.2: Visualization of Multi-Head Self-Attention mechanisms.

Linear Layer and SoftMax Function

In the final stage of the decoder, a linear layer receives the output from the preceding block and elevates it to a higher-dimensional space. By incorporating trainable parameters, this layer facilitates the acquisition of nuanced patterns and relationships within the data during the training process. Subsequently, the softmax function is employed on the output of the linear layer, converting raw scores into a probability distribution across the entire vocabulary of possible tokens.

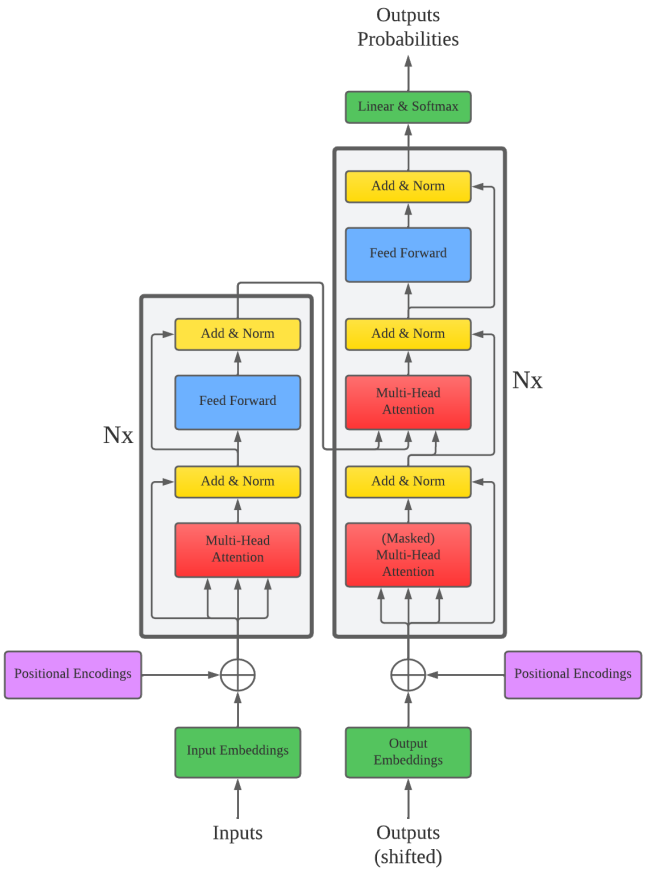


Figure 2.3: The Transformer - model architecture.

2.6.2 Use Cases

This section explores the use cases and architectures of key components within the Transformer framework, including the encoder, decoder, and the overall encoder-decoder structure.

- **Encoder Architecture** - The encoder architecture utilizes self-attention for tokens, capturing dependencies in input sequences through multiple stacked layers with residual connections and normalization. It excels in NLU tasks like sentiment analysis and named entity recognition. Prominent examples include models like BERT Devlin et al. (2018).
- **Decoder Architecture** - The decoder mirrors the encoder but includes a masked self-attention mechanism, making it suitable for NLG tasks like machine translation. Models like Generative Pre-Training Transformer (GPT) Radford and Narasimhan (2018) exemplify this architecture with their stack of Transformer decoder layers.
- **Encoder-Decoder Architecture** - The encoder-decoder architecture seamlessly integrates both components, facilitating hierarchical pattern recognition and catering to various NLP tasks such as Question Answering and Text Generation. Text-to-Text Transfer Transformer (T5) Raffel et al. (2019) is a notable instance.

2.6.3 Challenges with Standard Attention Mechanisms

Vanilla Transformer architecture heavily depends on the self-attention mechanism, which exhibits quadratic complexity in terms of memory and time. This complexity becomes apparent when processing lengthy sequences, imposing limitations on real-life applications. As articulated in Zhang and Jankowski (2022), challenges emerge prominently in tasks related to medical document understanding, notably efforts such as the assignment of International Classification of Disease (ICD) diagnosis codes, where the effective processing of lengthy medical documents is crucial. The computational cost of Transformers stems from factors like matrix multiplications and the introduction of feed-forward layers. These computational demands predominantly affect the training phase, requiring frequent weight updates and contrasting with the less demanding inference phase.

2.7 Long-Document Trasnformer

This section introduces the Longformer Beltagy et al. (2020), a modified Transformer architecture designed to address limitations related to the standard self-attention mechanism, particularly its quadratic complexity. This architecture reduces the model's complexity from $O(n^2)$ to $O(n \times w)$, through the implementation of a sparse attention matrix, represented in Figure 2.4, constrained by three primary fixed patterns:

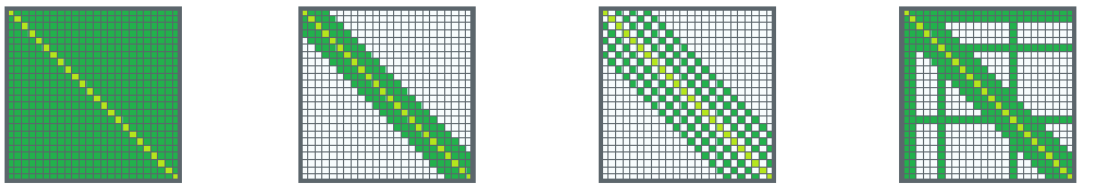
- **Global Attention** - Specific tokens within the sequence possess the ability to attend to all other tokens, resembling the conventional Transformer's attention mechanism. This operation is symmetric, allowing all other tokens to reciprocally attend to these specific tokens. Commonly, this is applied to special tokens such as [CLS], particularly in the context of sequence classification.
- **Local Attention or Sliding Window** - Recognizing the significant influence of local context, i.e., neighboring tokens, the architecture incorporates a fixed-size window attention around each token. This window, with a size of w attends to $\frac{w}{2}$ tokens on each side.
- **Dilated Sliding Window** - To augment the receptive range of tokens without imposing a substantial computational burden, the Longformer introduces a dilated sliding window. This window operates similarly to the sliding window but includes gaps of size d. The model's original implementation restricts the use of dilated sliding windows to higher layers, grounded in the rationale that lower layers should prioritize maximizing local context.

Structural Modifications and Pretraining

The Longformer model was pretrained by leveraging the Robustly Optimized BERT Pretraining Approach (RoBERTa) Liu et al. (2019) released checkpoint as a starting point. The authors introduced essential structural modifications to enable the model to accommodate fixed attention patterns. Specifically, they configured the window size w to 512 and expanded the maximum input size by extending the position embeddings from 512 to 4096. This extension was achieved by replicating the 512 positional embeddings of

RoBERTa multiple times. They also maintained the integrity of RoBERTa weights by freezing them during the training process, focusing solely on training the newly introduced position embeddings. The continuation of pretraining involved the use of a document corpus, employing the Masked Language Modeling (MLM) technique Devlin et al. (2018).

- **Masked Language Modeling** - A subset of tokens in the input sequence is randomly chosen and replaced with the special token [MASK]. The MLM objective encompasses a cross-entropy loss function, targeting the accurate prediction of the masked tokens.



(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

Figure 2.4: Comparasion between full self-attention pattern and Longformer's attention patterns.

2.8 Transfer Learning

Traditional ML models operate under the assumption that the test data conforms to the same distribution as the training data. Transfer Learning (TL) introduces flexibility by allowing variations in domains, tasks, and distributions between the training and testing phases. At its core, TL aims to distill knowledge from one or more source tasks and seamlessly apply it to a designated target task. This distinctive focus is illustrated in Figure 2.5, highlighting the process of applying knowledge obtained from a related source domain or learning task during the pretraining phase. Subsequently, this acquired knowledge is strategically employed to enhance the performance of the target predictive function within its domain during the fine-tuning phase.

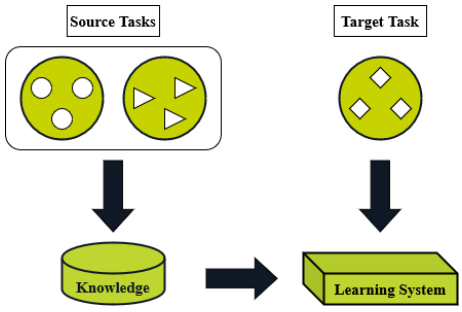


Figure 2.5: Learning process in Transfer Learning.

Drawing from the insights of Pan and Yang (2010), within a specific domain context denoted as D, formed by the feature space X, and the marginal probability distribution $P(X)$. Similarly, a specific task denoted as T, defined by the label space Y, and an objective function $f(.)$ inferred from training data. TL can be subdivided into three main categories:

- **Inductive Transfer Learning** - In scenarios where the source and target tasks diverge, with the target domain requiring labeling to induce a purposeful predictive model $f_t(.)$ for its unique domain.
- **Transductive Transfer Learning** - As articulated by Arnold et al. (2007), where the source and target tasks align, and domains may vary. Pan and Yang (2010) adds the requirement of part of the unlabeled target data to be seen at training time.
- **Unsupervised Transfer Learning** - Similar to Inductive Transfer Learning, but with a different yet related target task and unknown target labels.

Transformers have revolutionized NLP by enabling the use of Pre-Trained Models (PTMs), such as BERT and GPT. These models, pre-trained on extensive corpora, have learned powerful language representations that can be fine-tuned for specific tasks with minimal reliance on labeled data. This represents not only a technical breakthrough in model architecture but also a substantial operational advancement in training efficiency.

2.9 Knowledge Graphs

Knowledge bases (KBs) are structured repositories designed to store and retrieve information in both machine-readable and human-understandable formats. They serve as centralized sources of information, organizing facts, concepts, and relationships to facilitate efficient access and utilization. This thesis focuses on a particular type of KB known as a KG.

While there are many definitions of KGs, this thesis adopts a broad definition from Hogan et al. (2021): "a graph of data intended to accumulate and convey knowledge of the real world, whose nodes represent entities of interest and whose edges represent relations between these entities." In essence, a KG uses a graph-based data model to represent information, comprising nodes (representing entities) and edges (representing relationships between entities), as illustrated in Figure 2.6.

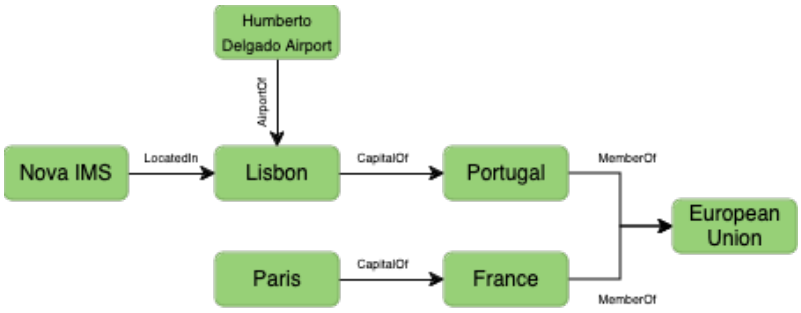


Figure 2.6: Example of a knowledge graph.

KG can be broadly categorized into two types: open knowledge graphs and enterprise knowledge graphs.

- **Open Knowledge Graphs:** These are publicly accessible knowledge graphs that are often created through collaborative efforts and published online for the benefit of the wider community. Examples include DBpedia Bizer et al. (2009), Wikidata Vrandečić and Krötzsch (2014), and YAGO Hoffart et al. (2011).
- **Enterprise Knowledge Graphs:** In contrast, enterprise knowledge graphs are typically developed and maintained within organizations for internal use. While not publicly accessible, major tech companies like Google, Bing, LinkedIn, and Facebook are known to utilize extensive enterprise knowledge graphs to power various aspects of their platforms Noy et al. (2019).

KGs serve as networks of semantic knowledge, storing interconnected descriptions of real-world entities and their relationships. Tian et al. (2022) formally defines a KG as a set of triplets: $K = (E, R, F)$. where:

- $E = \{e_1, e_2, \ldots, e_i\}$ is the set of entities or nodes within the graph. For instance, in Figure 2.6, both "Nova IMS" and "Lisbon" are members of the set E.
- $R = \{r_1, r_2, \ldots, r_j\}$ is the set of relationships or edges connecting these entities. In the Figure 2.6, "locatedIn" and "capitalOf" are examples of relationships within R.
- $F\{f_1, f_2, \ldots, f_k\}$ represents a fact, where each fact is defined by an head entiy e_h, a tail entity e_t and a relationship r. For example, the fact "Paris capitalOf France" is a triplet with "Paris" as the head entity, "capitalOf" as the relationship, and "France" as the tail entity.

Knowledge graphs excel at enhancing search and information retrieval by understanding the meaning behind queries and providing more relevant results. Some prominent examples of knowledge graphs that enhance search and information retrieval include:

- **Google's Knowledge Graph:** Powers Google Search, providing informative panels and direct answers to queries Singhal (2012).
- **Wolfram Alpha:** A computational knowledge engine that uses a massive KG to answer factual questions and perform complex calculations Cassel (2016).
- **Facebook's Social Graph:** Represents the network of connections between users, driving personalized recommendations and features Ugander et al. (2011).

2.10 Semantic Table Interpretation

Semantic Table Interpretation (STI), also known as Semantic Table Annotation, is a specialized field within NLP and Information Extraction (IE) that focuses on enriching tabular data with semantic meaning. This is achieved by annotating table elements (e.g., columns, cells) with concepts and relationships derived from a KG or database schema.

By establishing these connections, machines can effectively map tabular data into a KG, enabling them to intelligently process and derive valuable insights from the data.

Given a KG $K = (E, R, F)$, where E is the set of entities, R is the set of relations, T is the set of facts, and a table $T = (C_1, C_2, \ldots, C_n)$, where C_i is the i-th column of T and each column has a set of cell values $C_i = (v_{i1}, v_{i2}, \ldots, v_{im})$, where v_{ij} is the cell value in the j-th row of column C_i. Taking inspiration from Suhara et al. (2022), the three primary tasks of Semantic Table Interpretation STI are defined as follows:

- **Column Type Annotation (CTA):** Given a column $C_i \in T$ and a vocabulary of column types V_{cta} derived from E, the goal is to find the annotation $\hat{y}_i \in V_{cta}$ that best describes the semantic type of C_i. For example, in Figure 2.7, the column containing the names of football clubs would be annotated with the type "Association Football Club."

- **Cell Entity Annotation (CEA):** Given a cell value $v_{ij} \in C_i$ in column C_i and row j, and a vocabulary of entities V_{cea} derived from E, the goal is to link v_{ij} to the most appropriate entity $\hat{e}_{ij} \in V_{cea}$. For instance, "Arsenal" in a cell is linked to the KB entity representing "Arsenal F.C." (see Figure 2.7).

- **Column Property Annotation (CPA):** Given a pair of columns $(C_i, C_j) \in \mathcal{T}$, and a vocabulary of relations V_{cpa} derived from R, the goal is to identify the semantic relation $\hat{r}_{ij} \in V_{cpa}$ that best describes the relationship between C_i and C_j. For example, the relationship of "Home Venue" is identified between a column containing club names and the column listing their respective stadiums (see Figure 2.7).

According to Liu et al. (2023) the evolution of STI approaches can be categorized into three main paradigms:

- **Heuristic Methods:** These methods generate candidate entities from KG and then score and rank these entities based on metrics such as string similarity or TF-IDF. While effective in specific domains, heuristic methods may struggle with incomplete data or incompatibly between target tables and KG.

- **Feature-Engineering Based Methods:** This approach exploits statistical and lexical features extracted from tables, employing machine learning models for annotation. It requires labeled data and can be sensitive to feature quality.

- **Deep Learning Based Methods:** Deep learning models, particularly transformer architectures like BERT, have revolutionized STI by learning complex patterns and contextual representations without the need of hand-crafted features, often outperforming traditional methods. However, these models typically demand large amounts of training data. Recent advancements make use of pre-trained models and transfer learning to mitigate data limitations.

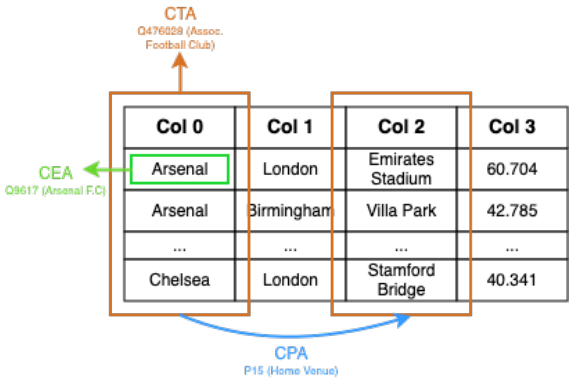


Figure 2.7: Illustration of the three main STI taks.

The ultimate goal of STI is to transform raw, potentially difficult-to-interpret tables into structured, machine-readable formats that can be easily integrated with existing KGs. By aligning table information with established entities and relationships within a KG, STI enhances the semantic richness and accessibility of the data, thereby enabling a wide array of downstream applications, as discussed in Section 2.9.

This thesis aims to contribute to the field by advancing the state-of-the-art in CTA, potentially leading the way for improvements in other STI tasks as well.

Chapter 3

Related Work

3.1 Word Embeddings

In the field of NLP, the advent of word embeddings marked a significant paradigm shift. Traditionally, text data, inherently categorical, posed a challenge for ML algorithms, which are designed for numerical input. Word embeddings address this by transforming words and phrases into continuous numerical vectors, bridging the gap between linguistic data and machine learning applications.

3.1.1 Traditional Word Embeddings

The concept of word embeddings has its roots in earlier models such as one-hot encoding Harris (1954), but these were limited by their inability to capture semantic relationships. The journey from these basic models to advanced ones like GloVe Pennington et al. (2014) reflects the field's progression toward more nuanced and effective language models. According to Almeida and Xexéo (2019), traditional word embeddings can be characterized as dense, distributed, and fixed-length vectors based on word co-occurrence statistics, aligning with the distributional hypothesis of language. These embeddings can be broadly categorized into two types:

Prediction-based Models

- **Word2Vec** - A neural network-based technique Mikolov et al. (2013), particularly its Skip-Gram and Continuous Bag of Words (CBOW) models, revolutionized the understanding of word contexts. Skip-Gram predicts context words from a target word, whereas CBOW predicts a target word from its context, both effectively creating an embedding matrix that captures the essence of word semantics.

Count-based Models

- **GloVe (Global Vectors for Word Representation)** - This model combines matrix factorization techniques with local context window methods, offering a in-depth perspective by analyzing word co-occurrence frequencies on a corpus-wide scale.

3.1.2 Contextual Word Embeddings

Contextual word embeddings mark a significant stride in the evolution of NLP, surpassing conventional models such as Word2Vec and GloVe. In contrast to their predecessors, these embeddings excel in capturing context-sensitive meanings, moving beyond static representations of words. This limitation is effectively addressed by approaches like Embeddings from Language Models (ELMo) Peters et al. (2018), which introduced a 2-layer LSTM encoder featuring a bidirectional language model, as well as subsequent innovations like OpenAI's GPT and BERT. These advancements further refine the understanding of language in context through dynamic and context-aware representations of words.

3.2 Tabular Language Models

The challenge of converting tabular data into numerical representations for machine learning parallels the development of word embeddings in NLP. However, unlike text, tables with their distinctive information presentation, well-organized numerical values, and diverse structures (e.g., relational tables, matrix tables, spreadsheets), introduce unique challenges. Initial approaches, like Deng et al. (2019), adapted the skip-gram neural network, treating rows of the table as sentences to create embeddings, setting a foundation for more sophisticated techniques. More recent works use modern PTMs based on transformer architectures to uncover the details of tabular data. Known as Tabular Language Models (TaLMs), built upon transformer architectures such as BERT and T5, excel in interpreting both the structural and semantic aspects of tabular data. Their effectiveness is notable, achieving high performance in various table-centric tasks, even without bespoke pre-training for tabular data. Adapting PTMs for tabular data involves a dual approach of data preprocessing and architectural modifications.

3.2.1 Data Preprocessing Techniques

These transformers, like their textual counterparts, require inputs in token sequences. Preprocessing is essential, with models like TABERT Yin et al. (2020) employing content snapshots for input size management, focusing on the most relevant table rows. Various serialization methods are adopted, depending on the model: row-wise flattening by DTR Herzig et al. (2021) and TURL Deng et al. (2022), column-wise by DODUO Suhara et al. (2022), and a combination of column and row flattening by TABBIE Iida et al. (2021). Context integration is another critical aspect, with models like TABERT and TAPAS Herzig et al. (2020) merging context through concatenation.

3.2.2 Architectural Modifications

The architectural changes in these PTMs are extensive, affecting input, internal, and output stages, and are tailored to meet the specific challenges posed by tabular data. At the input level, additional positional embeddings are often integrated to model table structures effectively. Models like TABERT and TAPAS use embeddings that represent the

position of the cell, indicated by its row and column IDs, to boost structural task performance. In contrast, TABLEFORMER Yang et al. (2022) utilizes per-cell positional embeddings to avoid row and column biases. Internally, modifications focus on structure-aware processing. TABERT, for example, uses vertical self-attention layers to capture cross-row cell dependencies. TURL employs a masked self-attention module focusing on elements in the same row or column, differing from the traditional all-element attention approach. At the output stage, extra layers may be added to the language model's feed-forward networks, customized for specific tasks such as TABFACT Chen et al. (2019) for Table Fact-Checking (TFC), DODUO for STI, and TAPAS for Table Question Answering (QA), necessitating the training of one or more additional layers.

3.2.3 Training-Procedure Modifications

In the training procedures for tabular data processing systems, two main strategies are employed: end-to-end training or fine-tuning after pre-training. These tasks typically focus on reconstruction, where systems aim to restore correct inputs from corrupted versions. Unlike traditional token-level MLM, these tasks are tailored to account for table structure. Common techniques involve masking tokens in both text and table cells, with variations like masked columns in TABERT and masked entities in TURL. In the majority of models the primary goal is to minimize cross-entropy loss in classification tasks, with some systems like GTR Wang et al. (2021b) using point-wise ranking objectives post-pre-training to rank tables based on query relevance.

3.2.4 Downstream Tasks of Tabular Language Models

Using neural representations for tabular data shows improvements in the performance of several downstream tasks. This subsection presents some of these tasks, shedding light on their inputs, outputs, and contextual relevance. While these tasks all center around tabular data, they exhibit diversity, with each task leveraging unique information, even within the same category.

- **Table Question Answering (QA):** This task revolves around extracting cell-based answers from tables in response to natural language questions. It ranges from straightforward lookup questions, as seen in models like TableQA Vakulenko and Savenkov (2017), DTR Herzig et al. (2021), and CLTR Pan et al. (2021), to more complex inquiries requiring advanced processing, exemplified by models such as TAPAS, MATE Eisenschlos et al. (2021), and TAPEX Liu et al. (2021).

- **Table-based Fact-Checking (TFC):** Shares similarities with Table QA but focuses on returning a Boolean value that represents whether a given hypothesis, presented as textual input, can be confirmed or refuted based on the information in a table. Models in this task include MATE, TAPEX, and TABFACT.

- **Semantic Parsing (SP):** Differs from Table QA by focusing on transforming natural language utterances into structured formats, such as SQL queries or textual descriptions. It includes tasks like:

  - **Table-To-Text Conversion:** Generating textual descriptions for a given table, with TABLEGPT Gong et al. (2020) as a notable model.
  - **Table-To-SQL Parsing:** Creating SQL queries based on a table and an NL question, with models like GRAPPA Yu et al. (2020) and TABERT proficient in this area.

- **Table Retrieval (TR):** Involves identifying and selecting relevant tables from a collection based on a given question. This task is crucial in narrowing down the search space for Table QA tasks. Notable models in this context include GTR and DTR, with a primary challenge being the management of the limited input size of transformers.

- **Table Content Population(TCP)/Table Augmentation:** TCP aims to accurately predict and fill in corrupted or missing cell content. Notable models in this task include TABBIE. Table Augmentation, while similar to TCP, specifically refers to the process of enriching a seed query table with additional data, involving models like TURL and KEN Cvetkov-Iliev et al. (2023).

3.3 Deep Learning Approaches in CTA

This subsection conducts a systematic survey of influential models in the domain of CTA, particularly with consideration for inter-table context. The models discussed, including DoDuo, RECA, TABBIE, and TURL, are presented within the framework of transformer architectures, showcasing unique approaches to handling table data through advanced neural network designs, more details on Table 3.1. Additionally, alternative strategies are explored through non-transformer models such as Sherlock, Sato, and TCN. This detailed overview aims to encapsulate the dynamic field of machine learning techniques applied to STI. Table 3.2 offers a concise summary of these models, serving as a focal point for understanding their key characteristics and contributions in the domain of structured data interpretation.

3.3.1 Models Using Transformer Architectures

- **DoDuo** - DoDuo Suhara et al. (2022) is a model that stands out for its approach to fine-tuning a 12-layer pre-trained BERT model. It utilizes a hard parameter-sharing multi-task learning approach to optimize its performance. In its approach to table serialization, DoDuo processes tables column by column, strategically inserting a [CLS] token at the beginning of each column to enhance contextual understanding. The model uses the [CLS] token embeddings, which are then processed through a custom-designed dense layer for CTA. DoDuo is acclaimed for its minimal input data requirement, efficiently utilizing as few as 8 tokens per column, showcasing remarkable data efficiency.

- **RECA** - The RECA framework Sun et al. (2023) is designed to enhance CTA, specifically focusing on addressing the challenges posed by wide tables. It starts by extracting named entities using tools like SpaCy and creates unique schema

strings for each table. RECA utilizes Jaccard similarity to filter and identify topically relevant candidate tables, further evaluating relatedness through edit distances of schema strings. RECA concatenates columns from the target, related, and subrelated tables to provide context, generating distinct input strings for BERT model processing. This results in detailed 768-dimensional embeddings, subsequently processed by a classification module to accurately annotate column semantic types.

- **TABBIE** - TABBIE Iida et al. (2021) introduces an innovative pretraining objective known as 'corrupt cell detection,' inspired by the ELECTRA framework. This method operates through a binary classifier on the final-layer cell representations to identify corrupted or replaced cells during preprocessing. TABBIE's unique table serialization process involves appending a [CLSCOL] token at the beginning of each column sequence and a [CLSROW] token at the start of row sequences, enhancing its understanding of table structures. The model utilizes Column and Row Transformers, both based on a pre-trained BERT model, to encode these serialized sequences. These Transformers work in tandem, averaging their outputs at each layer, which fosters a more contextualized representation of each cell within the table.

- **TURL** - TURL Deng et al. (2022) is a framework designed to understand complex relational tables on the web, with a specific focus on tasks such as predicting column types and column relationships. TURL transforms table inputs into a sequenced arrangement of tokens and entity cells, incorporating table metadata and scanning table content row by row. It utilizes a structure-aware Transformer encoder enhanced with masked self-attention and a custom-designed visibility matrix based on the table's inherent structure. TURL employs both the MLM approach by BERT and a pioneering pretraining objective named Masked Entity Recovery (MER). This objective involves masking input entity cells and subsequently recovering linked entities, relying on surrounding entity cells and contextual clues from table metadata.

Model	Uses Pre-Trained Model	Training Methodology
DoDuo	Yes	Fine-tunes pre-trained BERT model
RECA	Yes	Uses pre-trained BERT weights to initialize embeddings
TABBIE	No	Pre-trained using Corrupt Cell Detection, fine-tuned for specific tasks
TURL	No	Pre-trained using MLM and MER, fine-tuned for specific tasks

Table 3.1: Comparison of models using Transformer architectures.

3.3.2 Models Using Non-Transformer Architectures

- **Sherlock** - Sherlock Hulsebos et al. (2019) is a deep learning model designed for CTA. It employs various feature categories, including global statistics, character distributions, word embeddings, and paragraph vectors, to ensure fixed-length input representations. Sherlock utilizes a multi-input neural network trained on a substantial dataset, with distinct subnetworks for different feature categories. These subnetworks condense input features, and their outputs are combined with statistical features in the primary network for multiclass type detection classification.

- **Sato** - Sato Zhang et al. (2020) is a hybrid ML model that revolutionizes CTA by incorporating table context for enhanced accuracy. It consists of two innovative components. The first is a topic-aware prediction model employing Latent Dirichlet Allocation (LDA) for topic modeling, creating a topic vector that enhances a single-column model's, Sherlock is used in the paper, predictive capability. The second component addresses local column relationships with a structured output prediction model utilizing Unary and Pairwise potentials within a linear-chain Conditional Random Field (CRF) framework, enabling nuanced interpretation of semantic types based on both column content and inter-column relationships.
- **TCN** - TCN Wang et al. (2021a) is an advanced deep-learning framework tailored specifically for the complex task of interpreting web tables. It excels in learning detailed cell embeddings by aggregating contextual information within the table (intra-table) and across various tables (inter-table). TCN combines linear and non-linear transformations of each cell's column and row embeddings for intra-table context, capturing relationships within the table. Inter-table context involves three distinct modules: value aggregation, position aggregation, and topic aggregation, each contributing to the model's understanding of table content. TCN's training process, inspired by BERT's MLM, includes a pre-training phase for masked cell recovery and a supervised fine-tuning phase for accurate CTA.

Model	Transformer Based	Uses Inter-Table Context	Author
Doduo	Yes	No	Suhara et al. (2022)
RECA	Yes	Yes	Sun et al. (2023)
TABBIE	Yes	No	Iida et al. (2021)
TURL	Yes	No	Deng et al. (2022)
Sherlock	No	No	Hulsebos et al. (2019)
Sato	No	No	Zhang et al. (2020)
TCN	No	Yes	Wang et al. (2021a)

Table 3.2: Summary of models for CTA tasks.

Chapter 4

Methodology

4.1 Definitions

In this section, we introduce the notation used throughout the framework. Table 4.1 provides a detailed description of the notation employed, including symbols representing datasets, columns, serialized, embeddings, column values, ground truth labels, subsets of related columns, and string input results. This notation facilitates clarity and consistency in discussing various aspects of the methodology.

Table 4.1: Description of Notation.

Symbol	Description
D	Dataset containing n rows, each row represents a column to annotate
E	Dataset containing n rows, each row represents a column embedding
M_{type}	Column type prediction model
v_{type}	Pre-defined column semantic type vocabulary
$\|v_{type}\|$	Number of classes in v_{type}
C_i	The i-th column in D without header
v_i	The i-th value of a column
$S(C_i)$	Serialized representation of column C_i
E_i	Sentence embedding of column C_i
C_j^t	The j-th target column without header (column we aim to annotate)
y^t	Ground truth label, array with size $\|v_{type}\|$ for column C_j^t
$\hat{y}^t$	Predicted label, array with size $\|v_{type}\|$ for column C_j^t
H	The subsets of related columns for column C_j^t
R_k	The k-th subset of related columns
C_i^k	The i-th column of k-th subset
I_t	Input sequence resulting from $S(C_j^t)$
I_{R_k}	Input sequence resulting from concatenation of columns in R_k
$T(.)$	Apply tokenizer on a input sequence of stings
I_{token}	Tokenized input sequence prepared for the model
L_{attn}	Local attention mask, array with the same size of I_{token}
G_{attn}	Global attention mask, array with the same size of I_{token}
$\oplus$	String concatenation operation

4.2 Overview

This section presents the methodology for developing a Longformer-based framework to generate CTA from a designated KG. The entire process is depicted in Figure 4.1 and comprises three phases:

1. **Column Filtering:** For a target column C_j^t, we identify the N most similar columns within the dataset $D = \{C_1, C_2, \ldots, C_n\}$ using cosine similarity. The value of N is determined through empirical analysis. Subsequently, we create k subsets of related columns $H = \{R_1, R_2, \ldots, R_k\}$ where each subset contains columns with similar similarity scores.

2. **Data Preprocessing and Transformation:** We employ the Longformer tokenizer to convert the textual information in the target column C_j^t and the related column subsets $R_1, R_2, ..., R_k$ into numerical token IDs. If necessary, sequences are padded to a fixed length. Attention masks L_{attn} and G_{attn} are also created to guide the model's focus during training.

3. **Classification and Column Annotation:** The Longformer model processes the input sequence I_{token} along with the attention mask L_{attn} and global attention mask G_{attn}. The output of the Longformer is then fed into a multi-class classification head, which predicts the column type annotation y^t for C_j^t from the pre-defined vocabulary v_{type}.

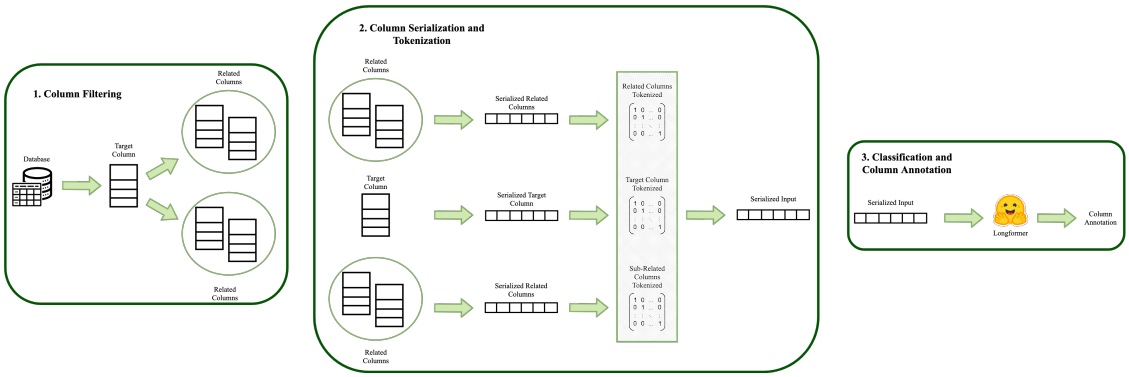


Figure 4.1: Overview of the methodology.

4.3 Column Filtering

To integrate inter-table context, we search for columns that exhibit semantic similarities with the target column C_j^t. We utilize sentence transformers from the Sentence Transformers library[1] for this task, treating the column values within each column as input to the model. We choose Sentence Transformers (Reimers and Gurevych (2019)) over traditional transformer models because they are specifically designed for semantic similarity

[1] Sentence Transformers Library.

tasks, generating embeddings that capture the meaning of text at the sentence level. Algorithm 3 outlines the systematic process of converting the raw dataset D into a dataset of column embeddings E.

Subsequently, we assess the semantic similarity between C_j^t (the target column) and all other columns C_i for $j \neq i$ using the cosine similarity formula:

$$\text{Similarity}(E_j^t, E_i) = \frac{\sum_{k=1}^{d} E_{j_k}^t \cdot E_{i_k}}{\sqrt{\sum_{k=1}^{d} (E_{j_k}^t)^2} \sqrt{\sum_{k=1}^{d} (E_{i_k})^2}} \tag{4.1}$$

where E_t denotes the embedding of the target column C_j^t, E_i is the embedding of another column C_i, and d represents the dimensionality of the embedding.

This computation yields an array with dimensions corresponding to the number of columns in the dataset n. We then select the top N most similar columns based on their cosine similarity scores. These selected columns are divided into $k = \frac{N}{4}$ subsets $H = \{R_0, R_1, ..., R_{k-1}\}$, where each subset contains 4 columns with similar similarity scores. This process is represented in Algorithm 4.

4.4 Data Preprocessing and Transformation

Once we have H, the next step involves transforming the data into a format suitable for tokenization. We achieve this by serializing each column. Suppose a column C_i has column values $(v_1, v_2, \ldots, v_m)$ as strings; the serialized representation of the column S_i is given by:

$$S(C_i) = v_1 \oplus v_2 \oplus \ldots \oplus v_m \tag{4.2}$$

To obtain the input string for the target column I_t, we apply Equation 4.2 on C_j^t. For each subset of related columns we concatenate the serialized values of each column in the respective subset and obtain the strings I_{R_K}:

$$I_{R_k} = S(C_1^k) \oplus S(C_2^k) \oplus \ldots \oplus S(C_N^k) \tag{4.3}$$

We utilize the Longformer tokenizer, denoted as $T(\cdot)$, to convert the input strings into numerical token IDs. Each input string is first prepended with a [CLS] token and appended with a [SEP] token. The target column's input string, I_t, is tokenized with a maximum length of 256 tokens, while each input string representing a related column subset, I_{R_k}, is tokenized with a maximum of 64 tokens per related column. This tokenization process ensures that no individual input string exceeds half the Longformer's local window size, *w*, thus maintaining contextual relevance within each local attention window.

The tokenized sequences are then combined into a single input sequence I_{token}, padded with [PAD] tokens to ensure all sequences have the same length. Corresponding attention masks L_{attn} and G_{attn} are also created. Given the local window size *w* of the Longformer's pretrained model[2] is 512, each token interacts with the preceding and succeeding 256 tokens to maintain contextual relevance. The sequence input formation proceeds as follows:

[2]Longformer Checkpoint.

$$I_{\text{token}} = T(I_t) \oplus ([\text{PAD}] \times 256) \oplus T(I_{R_1}) \oplus ([\text{PAD}] \times 256) \oplus \ldots T(I_{R_k}) \tag{4.4}$$

The global attention mask, G_{attn}, is a binary array adhering to the Longformer documentation. It governs the application of both local and global attention mechanisms. Following the library's guidelines, a value of 0 activates local (sliding window) attention, while a value of 1 activates global attention. Specifically, we focus global attention on the [CLS] tokens, as illustrated in Figure 4.2.

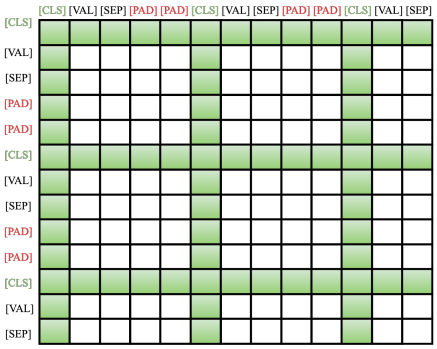
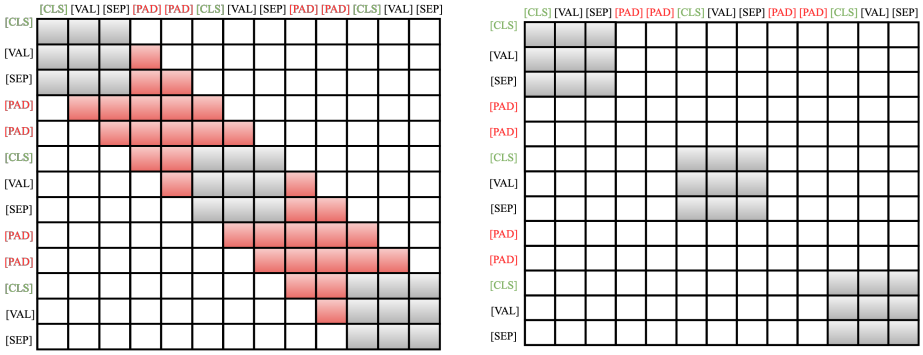


Figure 4.2: Global attention mechanism for *w*=4.

Likewise, the local attention mask, L_{attn}, is a binary array designed to prevent attention calculations on padding tokens [PAD]. This is achieved by assigning 0 to padding tokens and 1 to the remaining tokens. As depicted in Figure 4.3, the padding tokens are effectively masked from both local and global attention, serving solely to delimit the input sequences for local attention.



a) Including padding tokens b) Excluding padding tokens

Figure 4.3: Local attention mechanism for *w*=4.

The process of serialization and tokenization is detailed in Algorithm 6.

4.5 Classification and Column Annotation

With the preprocessed input sequence I_{token}, local attention mask L_{attn}, and global attention mask G_{attn} prepared, we feed them into the Longformer model M_{type}. The Long-

former's encoder processes this input, capturing both local and global contextual relationships within and across the target column and its related subsets.

The output of the encoder is then passed through a custom pooler layer. This layer extracts the final hidden state (a 768-dimensional vector) corresponding to each of the $k + 1$ [CLS] tokens. These extracted embeddings serve as rich representations of the target column and the k related column subsets, encapsulating the semantic information captured by the Longformer.

To mitigate potential overfitting, a dropout layer is applied to the extracted [CLS] embeddings. This layer randomly sets a fraction of the activations to zero during training, encouraging the model to learn more robust representations.

Next, the $k + 1$ [CLS] embeddings are fed into the classifier layer. In our initial architecture, each [CLS] token was processed by a separate dense layer with 768 input units and $|v_{\text{type}}|$ output units (corresponding to the number of classes in the predefined vocabulary v_{type}. These dense layers produced class probability distributions denoted as $(y_t, y_1, ..., y_k)$, where y_t corresponds to the target column and y_1 to y_k correspond to the related subsets. This process is illustrated in Figure 4.4.

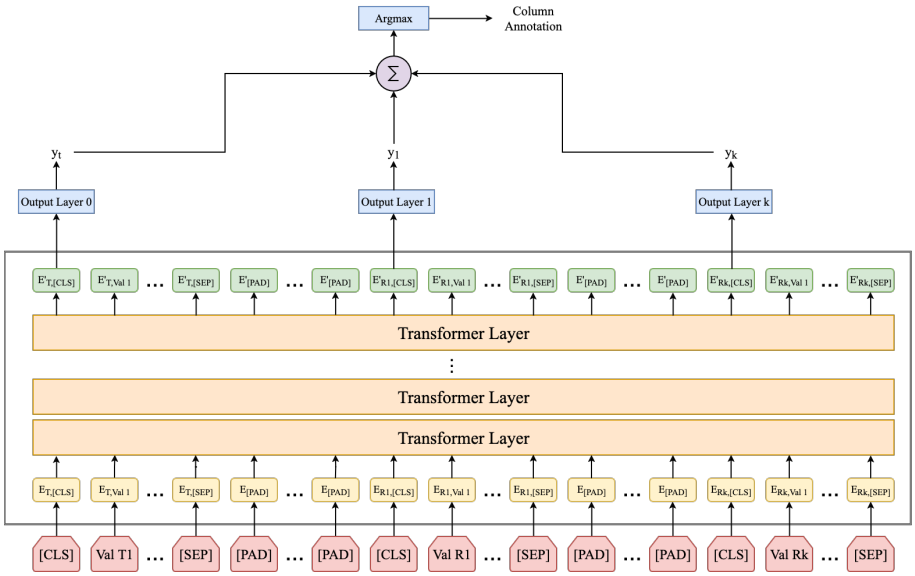


Figure 4.4: Overview of the CTA task.

However, we also explored a simplified architecture with only dense layers: one for the target column and one shared across all related subsets. This significantly reduces the number of parameters in the classifier while maintaining comparable performance. As shown in Figure 4.5

Regardless of the configuration, the outputs of the classifier heads are then aggregated to form a final prediction($\hat{y}$:

$$\hat{y} = \alpha y_t + \beta y_1 + \ldots + \gamma y_k \tag{4.5}$$

where α, β, and γ are trainable sets of dense layer weights. We apply an argmax function to $\hat{y}$ to select the class with the highest probability, which becomes the final column type annotation.

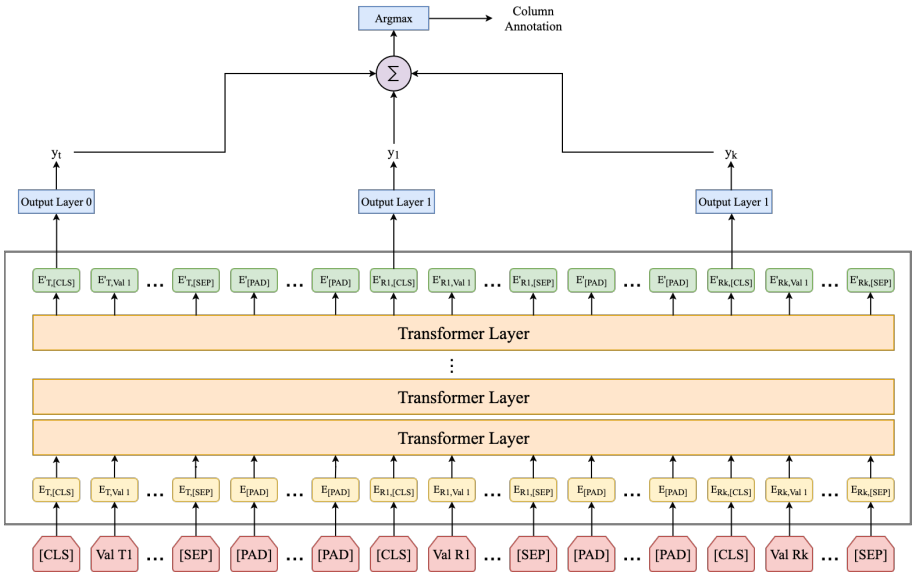


Figure 4.5: Overview of the modified classifier configuration.

4.6 Evaluation and Experimental Setup

4.6.1 Dataset

Our research utilizes the Schema.org Table Annotation Benchmark (SOTAB) (Korini et al. (2022)), chosen specifically for its suitability in addressing the challenges of large and diverse tables. SOTAB is specifically designed for STI tasks and contains approximately 50,000 annotated tables sourced from various websites, encompassing Schema.org (Guha et al. (2016)) data from different websites. It covers diverse entity types (events, businesses, products) and features 91 CTA labels (textual, numerical, date/time). A key factor in selecting SOTAB is its high median number of columns per table and the large number of samples it contains. This diversity is crucial for evaluating our Longformer-based model, which is designed to excel at handling wide tables and capturing contextual information across multiple columns. Table 4.2 summarizes SOTAB and other existing benchmarks, showcasing their size (number of tables/rows/columns) and the number of labels for CTA task.

Table 4.2: Overview of existing CTA benchmarks.

Benchmark	**Tables**	**Median**		**CTA**	
		Rows	**Columns**	**Columns**	**Labels**
Hard Tables Jiménez-Ruiz et al. (2020)	8,957	7	2	9,398	2,235
GitTables Hulsebos et al. (2023)	1,101	25	11	721/2,533	59/122
2T Cutrona et al. (2020)	180	89	4	540	39/276
BioDiv Abdelmageed et al. (2021)	50	99	17	614	92
WikiTable Deng et al. (2022)	580,171	8	5	654,670	255
SOTAB Korini et al. (2022)	107,927	42	8	162,351	91

Figure 4.6 illustrates examples of CTA from SOTAB, the first column is labeled ”Ho-

tel/name" as it contains hotel names. Other CTA labels include "addressLocation," "addressLocality," "Country," and "currency."

We adhere to the predefined splits in SOTAB for fair comparison with other models. Table 4.3 summarizes the number of tables, columns, and median label count per task for each split.

Hotel/name	streetAddress	addressLocality	Country	currency
Lau's Gateway	209 Main Street	Alofi	NU	NZD
Radisson Blu Hotel, Nice	223 Promenade Des Anglais	Nice	FR	EUR
Phoenix Park Hotel	38-39 Parkgate Street Dublin 8	Dublin	IE	EUR

Figure 4.6: Example of table annotation. The CTA labels are positioned atop each column, indicating their respective column types.

Table 4.3: SOTAB dataset split for CTA task.

Task	Training Set		Small Training Set		Validation Set		Test Set	
	#Tables	#Columns	#Tables	#Columns	#Tables	#Columns	#Tables	#Columns
CTA	46,790	130,471	11,517	33,004	5,732	16,840	7,026	15,040

The SOTAB dataset provides several subsets of the test set to evaluate how well a model handles specific annotation challenges:

- **Missing Values (MV):** Contains columns with 10-70% of their rows filled with values, assessing the model's ability to handle sparse data. This subset includes 2,808 columns annotated with 66 different CTA labels.
- **Value Format Heterogeneity (FH):** Includes columns with the same label but different value formats (e.g., date formats, measurement metrics), challenging the model to recognize semantic similarity despite variations in representation. This subset contains 619 columns annotated with 10 CTA labels.
- **Corner Cases (CC):** Focuses on columns that are inherently difficult to annotate due to their similarity to columns with different labels or their deviation from typical values within their own label. These columns were identified using TF-IDF, cosine similarity, and Kolmogorov-Smirnov tests. This subset includes 3,015 columns annotated with 59 different CTA labels.
- **Random Columns (RC):** Serves as a baseline for comparison, containing a random selection of columns for each label, with 8,598 columns annotated with 91 different labels.

Each subset contains at least 10 examples per label, ensuring a minimum level of representation for each category. By evaluating the model's performance on these diverse subsets, we can gain a deeper understanding of its strengths and weaknesses in handling various real-world challenges.

4.6.2 Model Overview

Our framework builds upon the Longformer transformer, retaining its positional embeddings and encoder. We customize the pooler layer to extract the final hidden states of $k + 1$ [CLS] tokens without additional parameters. For the CTA task, we add a classifier with individual dense layers for each [CLS] token, resulting in the model architecture detailed in Table 4.4. We choose the Longformer due to its ability to handle long sequences, essential for capturing context across multiple columns. The pooler layer modification allows for efficient extraction of relevant information for classification.

Table 4.4: Number of Parameters in Model.

Layer Name	**Number of Parameters**
Positional Embeddings	$41,753,088$
Encoder Layer	$8,859,648 \times (12)$
Pooler Layer	$590,592$
Classifier Dense Layer	$69,888 \times (k+1)$

4.6.3 Training Procedure

We train our model using the benchmark splits and an AdamW optimizer with a learning rate scheduler that linearly decreases from the initial value to 0 after a warmup period. Hyperparameter optimization is performed for the initial learning rate, warmup ratio, and accumulation steps, with results presented later. Model checkpoints are saved regularly. Due to the benchmark's constraint of a single annotation per column for CTA, we use cross-entropy loss for multiclass classification.

$$\mathcal{L}(y, \hat{y}) = -\sum_{j=1}^{C} y_j \log(\hat{y}_j) \tag{4.6}$$

The Formula 4.6 is for one sample, where C is the number of classes, y_j is a binary indicator (0 or 1) of whether class j is the correct classification for the sample, $\hat{y}_j$ is the predicted probability that the sample belongs to class j. It takes as input the ground truth label y^t array and the predicted label $\hat{y}^t$ array, respectively.

4.6.4 Evaluation Methodology

We evaluate our models using macro, weighted, and micro F1 scores, providing a broad assessment of their performance. We compare our results against a Random Forest classifier, TURL, and Doduo, adhering to the SOTAB paper's methodology, which only reports micro-averaged F1. We also report macro and weighted F1 to provide a more nuanced evaluation, as micro F1 can be misleading when class distributions are imbalanced.

$$\text{Precision}_i = \frac{TP}{TP + FP} \tag{4.7}$$

$$\text{Recall}_i = \frac{TP}{TP + FN} \tag{4.8}$$

The F1 score for i-th class is then computed as:

$$F1_i = \frac{2 \times \text{Precision}_i \times \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i} \tag{4.9}$$

In the multiclass classification setting we need to average the f1 scores from the various classes. In our work, we use macro-averaged F1, and weighted-averaged F1.

$$\text{Macro F1} = \frac{1}{C} \sum_{i=1}^{C} F1_i \tag{4.10}$$

$$\text{Weighted F1} = \sum_{i=1}^{C} w_i \times F1_i \tag{4.11}$$

The macro-averaged is computed using 4.10 by taking the arithmetic mean of all the per-class F1 scores, where C is the number of classes. The weighted-averaged is calculated using 4.11 by taking the mean of all per-class F1 scores, while considering the number of occurrences w_i of the i-th class in the dataset D. We also include the micro-averaged F1 that computes a global average F1 score by counting the sums of the True Positives (TP), False Negatives (FN), and False Positives (FP).

$$\text{Micro Precision} = \frac{\sum_{i=1}^{C} TP_i}{\sum_{i=1}^{C} (TP_i + FP_i)} \tag{4.12}$$

$$\text{Micro Recall} = \frac{\sum_{i=1}^{C} TP_i}{\sum_{i=1}^{C} (TP_i + FN_i)} \tag{4.13}$$

$$\text{Micro F1} = \frac{2 \times \text{Micro Precision} \times \text{Micro Recall}}{\text{Micro Precision} + \text{Micro Recall}} \tag{4.14}$$

Aligned with the SOTAB paper's methodology, we benchmarked our model against three supervised methods: a simple Random Forest classifier using TF-IDF-weighted words as features, TURL, and Doduo. TURL was fine-tuned for 50 epochs, while Doduo underwent 30 epochs of fine-tuning, both using a learning rate of 5×10^{-5}.

4.6.5 Hardware

We utilize a pre-trained Longformer checkpoint for environmental reasons and implement our framework using Sentence Transformers[1], Faiss[3], PyTorch[4], and Hugging Face Transformers[5]. Experiments were conducted on one NVIDIA TITAN V 32GB with CUDA 11.4.

[3]Faiss Library.
[4]PyTorch Library.
[5]Hugging Face Transformers.

Chapter 5

Training and Inference Framework

Having established the methodology in the previous chapter, we now dive into the heart of our framework: the training and inference of the Longformer-based model for CTA. This chapter details the algorithms that drive this process, showcasing the model's ability to learn relationships within column subsets during training and then take advantage of this knowledge to draw relationships between these subsets and unseen columns during inference, ultimately leading to accurate column type annotations. The pseudocode for the algorithms referenced in this chapter can be found in Appendix A.

5.1 Fine-Tuning Procedure

In the fine-tuning phase (Algorithm 1), the model learns relationships between serialized column representations and their corresponding column types, as well as the relationships among columns within the related subsets. The goal is to produce a trained model M_{type} capable of accurately predicting column types in new, unseen data.

The algorithm commences by generating column embeddings (Line 4) for both the training and validation datasets using the approach detailed in Algorithm 3. For each column C_i in the training set, the algorithm (Lines 8 - 13) executes two key steps:

1. Using Faiss[3], it semantically searches for the top-(N) related columns with similar embeddings to C_i, forming the related subsets H_{train}.

2. It applies a preprocessing step (Algorithm 6) to C_i and its related columns in H_{train}.

The same preprocessing is then applied to the validation dataset (Lines 15 - 20).

The algorithm then enters a training loop with a predetermined number of steps. In each step, a batch of data from the training dataset is fed into the model M_{type}. This data includes the tokenized input sequence I_{token}, attention masks L_{attn} and G_{attn}, and the ground truth label y_i. The model calculates a training loss, which is then used to update its weights through backpropagation (Line 28).

Periodically, the model is evaluated on the validation dataset (Line 29). During evaluation, the model generates predictions, computes a validation loss, and calculates relevant metrics (Line 38) to assess its performance. If predefined criteria are met, the training loop may be terminated early.

Upon completion of the training loop, the algorithm returns the fine-tuned model M_{type}, ready for deployment in CTA tasks on new, unseen data.

Algorithm 1 Training Procedure

Input: Raw training dataset C_{train}, raw validation dataset C_{val}, number of related columns N, total training steps tot_steps and evaluation steps $eval\_steps$
Output: Column type prediction model M_{type}

// Step 1: Create column embeddings
$E_{\text{train}} \leftarrow$ Alg. $3(C_{\text{train}})$
$E_{\text{val}} \leftarrow$ Alg. $3(C_{\text{val}})$

// Step 2: Train Dataset Preprocessing
$D_{\text{train}} \leftarrow$ Empty dataset
for each C_i^t **in** C_{train} **do**
    $H_{\text{train}} \leftarrow$ Alg. $4(E_{\text{train},i}, E_{\text{train}}, C_{\text{train}}, N)$
    $D_{\text{train}}.append($Alg. $6(C_i^t, H_{\text{train}}), y_i^t)$
end for

// Step 3: Validation Dataset Preprocessing
$D_{\text{val}} \leftarrow$ Empty dataset
for each C_j^t **in** C_{val} **do**
    $H_{\text{val}} \leftarrow$ Alg. $4(E_{\text{val},j}, E_{\text{train}}, C_{\text{train}}, N)$
    $D_{\text{val}}.append($Alg. $6(C_j^t, H_{\text{val}}), y_j^t)$
end for

// Step 4: Training Loop
for $step = 1$ **to** tot_steps **do**
    // Do Training
    for $input_i$ **in** D_{train} **do**
        $I_{\text{token}}, L_{\text{attn}}, G_{\text{attn}}, y_i^t \leftarrow input_i$
        $training\_loss_i \leftarrow M_{\text{type}}(I_{\text{token}}, L_{\text{attn}}, G_{\text{attn}}, y_i^t)$
        UpdateModelWeights($training\_loss_i$)
        if $step \mod eval\_steps == 0$ **then**
            break
        end if
    end for
    // Do Evaluation
    for $input_j$ **in** D_{val} **do**
        $I_{\text{token}}, L_{\text{attn}}, G_{\text{attn}}, y_j^t \leftarrow input_j$
        $logits_j, validation\_loss_j \leftarrow M_{\text{type}}(I_{\text{token}}, L_{\text{attn}}, G_{\text{attn}}, y_j^t)$
        $\hat{y}^t \leftarrow$ argmax$(logits_j)$
        ComputeMetrics$(\hat{y}^t, y_j^t)$
    end for
end for

return M_{type}

5.2 Inference Procedure

In the inference phase (Algorithm 2), the fine-tuned model M_{type} is employed to predict column types for new, unseen columns. This process utilizes the column embeddings derived from the training data E_{train} to identify semantically similar columns in the test set (Line 10), effectively transferring knowledge learned during training.

For each column in the test dataset, the algorithm identifies related columns based on their semantic similarity, forming subsets of related columns (Line 10). This step is crucial as it allows the model to draw upon the relationships learned during training to make inferences on new data.

Next, the target column and its related columns subsets undergo preprocessing (Line 11), resulting in the tokenized input sequence I_{token} and corresponding attention masks L_{attn} and G_{attn}.

The preprocessed input is then fed into the trained model M_{type}, generating predictions for the target column. These predictions are typically represented as logits (Line 18) for each column type in the vocabulary. The argmax function is applied to select the class with the highest probability, yielding the final predicted column type (Line 19) .

Finally, the algorithm iterates through all columns in the test dataset, repeating the above steps, and returns a list containing the predicted column types for each column.

Algorithm 2 Inference Procedure

1: **Input:** Embeddings training dataset E_{train}, raw test dataset C_{test}, number of related columns N and fine-tuned column type prediction model M_{type}
2: **Output:** List of predictions $preds$
3:
4: **// Step 1: Create column embeddings**
5: $E_{\text{test}} \leftarrow$ Alg. $3(C_{\text{test}})$
6:
7: **// Step 2: Test Dataset Preprocessing**
8: $D_{\text{test}} \leftarrow$ Empty dataset
9: **for** each C_j^t in C_{val} **do**
10: $\quad H_{\text{test}} \leftarrow$ Alg. $4(E_{\text{test},j}, E_{\text{train}}, C_{\text{train}}, N)$
11: $\quad D_{\text{test}}.append($Alg. $6(C_j^t, H_{\text{test}}), y_j^t)$
12: **end for**
13:
14: **// Step 3: Inference Loop**
15: $preds \leftarrow$ Empty predictions list
16: **for** $input_i$ **in** D_{test} **do**
17: $\quad I_{\text{token}}, L_{\text{attn}}, G_{\text{attn}}, y_i^t \leftarrow input_i$
18: $\quad logits_j \leftarrow M_{\text{type}}(I_{\text{token}}, L_{\text{attn}}, G_{\text{attn}}, y_j^t)$
19: $\quad \hat{y}^t \leftarrow \text{argmax}(logits_j)$
20: $\quad preds.append(\hat{y}^t)$
21: **end for**
22:
23: **return** $preds$

Chapter 6

Analysis of the Results

In this chapter, we rigorously evaluate the effectiveness of our proposed CTA framework, which was introduced in Chapter 2. We conduct an analysis of various factors that influence the model's performance, aiming to provide a nuanced understanding of its strengths, limitations, and potential for real-world applications. We present detailed experimental results, statistical analyses, and visualizations to support our findings and conclusions, with complete training and evaluation loss curves provided in Appendix B. The key areas of investigation in this chapter are as follows:

- **Column Embeddings Evaluation:** We assess the impact of different pre-trained sentence transformer models on the quality of column embeddings, a crucial component in identifying semantically similar columns.
- **Tokenizer Maximum Length:** We investigate the optimal maximum length for the Longformer tokenizer, considering the trade-off between capturing richer contextual information and maintaining computational efficiency.
- **Training Strategies:** From Scratch vs. Fine-Tuning: We examine the trade-offs between training our Longformer-based model from scratch and fine-tuning a pre-trained model, considering factors such as performance, convergence speed, and resource requirements.
- **Impact of Related Subsets:** We analyze the effect of varying the number of related column subsets k on model performance. This investigation aims to identify the optimal number of subsets that maintains a balance between providing sufficient contextual information and avoiding computational overhead.
- **Handling Special Cases:** We evaluate the framework's robustness and adaptability by assessing its performance on various challenging test cases, including columns with missing values, varying formats, and corner cases.
- **Influence of Dataset Size:** We investigate how the size of the training dataset affects the model's performance, providing insights into the relationship between data availability and annotation accuracy.

6.1 Evaluating the Impact of Sentence Transformer Models on Column Embeddings

We utilize sentence transformers to generate semantic embeddings for the columns to be annotated, facilitating the identification of semantically similar columns as detailed in Section 2.3. To assess the quality of these embeddings, we evaluate four pre-trained sentence transformer models (Table 6.1) on the validation set using two custom metrics:

- **SCP (Similar Columns Percentage):** The percentage of columns among the most similar ones that share the same label as the target column.
- **ZRC (Zero Related Columns):** The percentage of samples where none of the most similar columns share the same label as the target column.

Model	Dimensions	Encoding Speed	Model Size
all-mpnet-base-v2	768	2800	420 MB
all-distilroberta-v1	768	4000	290 MB
all-MiniLM-L12-v2	384	7500	120 MB
all-MiniLM-L6-v2	384	14200	80 MB

Table 6.1: Pre-trained models for semantic similarity.

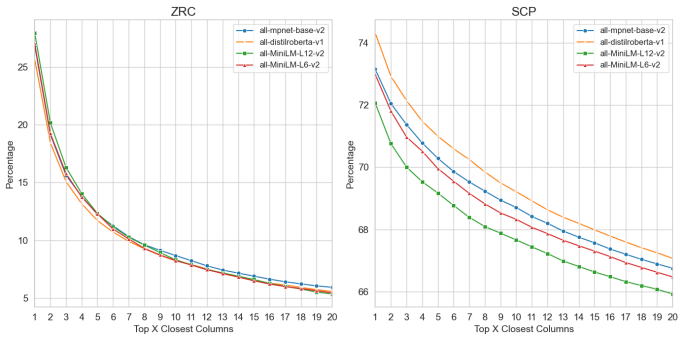


Figure 6.1: Column embeddings evaluation.

The results of this evaluation are presented in Figure 6.1. Based on this evaluation, we select the *all-distilroberta-v1* model, as it demonstrates the highest SCP while maintaining a ZRC comparable to other models. This suggests it keeps a good balance between retrieving semantically similar columns and avoiding unrelated ones.

Additionally, Figure 6.1 illustrates a trend: increasing the number of related columns decreases ZRC (fewer cases with no related columns), but then SCP also decreases as more dissimilar columns are included, indicating a decline in the overall quality of the related column sets.

6.2 Tokenizer Maximum Length

After selecting the sentence transformer model, we focus on tuning one crucial hyperparameter: the maximum length of the Longformer tokenizer max_length. A larger max_length allows for more tokens per column, providing richer contextual information, but reduces the number of related columns that can be included in each subset due to the maximum token limit per set (256 tokens). To investigate this trade-off, we conducted an experiment where all other hyperparameters were held constant and the model was trained from scratch for 3 epochs (12000 steps) with batch size 8 and accumulation steps 4. We tested max_length values of 32 and 64 tokens for related subsets and 256 tokens for target column, as shown in Figure 6.2.

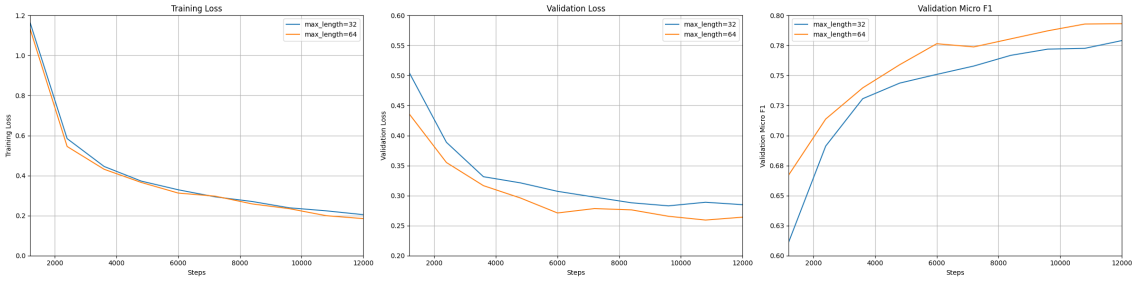


Figure 6.2: Impact of max_length on training and validation loss, and micro-F1 score.

As the number of tokens per column increases, the training loss remains relatively stable. However, both the validation loss and micro-F1 score improve significantly when using 64 tokens. This suggests that using fewer related columns but encoding each column with more tokens allows the model to capture more fine-grained semantic information, facilitating better generalization to unseen data. Therefore, all subsequent experiments were conducted using a max_length of 64 tokens per column for related subsets and 256 tokens for target column.

6.3 Comparing Training Strategies: From Scratch vs. Fine-Tuning

As explained in Section 2.8 Fine-tuning is a powerful technique in DL that takes advantage of knowledge acquired from a pre-trained model[2] on a related task. This section examines the effectiveness of fine-tuning compared to training from scratch for our Longformer-based column type annotation model. We investigate whether the pre-trained model's knowledge can be successfully transferred to our specific task, leading to improved performance and faster convergence. To evaluate this, we conduct four experiments as summarized in Table 6.2.

The learning rate and batch size for the experiments on the full dataset were selected based on the Longformer paper (Beltagy et al. (2020)), which demonstrated their effectiveness for text classification tasks in the fine-tuning setting. For the small dataset, we maintained the same number of total training steps as the full dataset experiments to ensure a fair comparison. This allowed us to observe the impact of fine-tuning with a smaller amount of training data while keeping the overall training effort consistent. We

Table 6.2: Experiments comparing training from scratch vs. fine-tuning for column type annotation.

Training Method	k	Epochs	Dataset	Batch Size	Accumulation Steps	Learning Rate
From Scratch	2	4	Full	8	4	3e-5
Fine-tuning	2	3	Full	8	4	3e-5
From Scratch	2	4	Small	8	1	1e-5
Fine-tuning	2	3	Small	8	1	1e-5

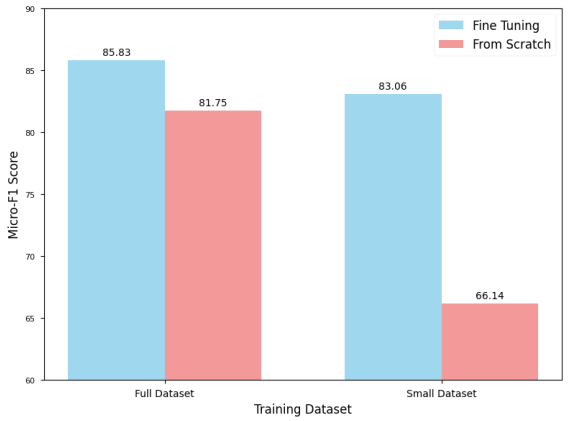


Figure 6.3: Micro-F1 Scores for CTA Experiments on Validation Dataset.

also reduced the learning rate for the small dataset experiments to avoid overfitting. For all experiments we used 5% of the total steps as warmup.

As shown in Figure 6.3, fine-tuning clearly outperforms training from scratch, yielding a +16.9% improvement in micro-F1 score using the small dataset for training and a +4.1% improvement using the full dataset. This demonstrates the effectiveness of transfer learning, where the pre-trained model's knowledge learned from MLM is successfully transferred to the CTA task.

6.4 Finding the Number of Related Subsets

In this section, we investigate the impact of the number of related subsets k on the performance of our Longformer-based model. As a reminder, these related subsets provide additional contextual information to the model for each target column. A higher value of k could potentially improve the model's understanding of relationships between columns, but it also increases the input size and computational complexity.

To identify the optimal number of related subsets k, we systematically varied this hyperparameter while keeping other model settings constant. We systematically vary k from 1 to 4, keeping other hyperparameters constant. Specifically, we fix the learning rate at 3e-5, warmup ratio at 5%, and batch size at 8. Due to memory constraints with our chosen batch size, a k value of 5 would result in an input sequence that exceeds the Longformer model's maximum capacity for this batch size. The accumulation steps are set to 4 for the full dataset and 1 for the small dataset, adjusting for the differences in dataset size. For each experiment, we fine-tune the pretrained Longformer model using

the corresponding hyperparameters.

Table 6.3: Performance on the validation and test sets using various k values and dataset sizes for training.

k	Dataset	Validation F1	Test F1						
		Micro	Micro	Macro	Weighted	MV	CC	FH	RC
1	Full	85.41	82.14	82.35	81.82	83.76	75.35	93.21	83.19
1	Small	82.43	78.42	77.47	77.62	80.38	72.90	90.31	78.91
2	Full	85.83	82.45	82.30	82.12	83.80	75.78	92.73	83.61
2	Small	83.06	78.83	78.06	78.08	81.16	72.60	92.08	79.30
3	Full	85.93	82.63	82.71	82.32	83.72	75.82	93.37	83.89
3	Small	82.90	78.91	77.92	78.14	81.05	72.27	91.28	79.65
4	Full	86.08	82.53	82.72	82.25	83.94	75.92	93.05	83.63
4	Small	83.25	79.01	78.04	78.24	80.59	73.20	91.44	79.65

To determine the optimal number of related subsets k, we conducted a repeated measures ANOVA on the F1 scores from Table 6.3. This statistical test is appropriate because the same model was evaluated on the same datasets under different conditions k values, making the observations dependent.

Before proceeding with the ANOVA, we rigorously assessed the necessary assumptions. Mauchly's Test of Sphericity yielded p-values of 0.364 and 0.295 for the small and full datasets, respectively. These values exceed the commonly chosen significance level of 0.05, indicating that the assumption of sphericity was not violated. Furthermore, Shapiro-Wilk tests for normality on each k value for both datasets resulted in p-values consistently above 0.05. This confirms that the data adheres to the normality assumption. Having met these prerequisites, we proceeded with the repeated measures ANOVA to investigate the impact of k on model performance.

Table 6.4: Post-Hoc Tests for Different Training Datasets.

H_1	Large Training Set		Small Training Set	
	Corrected p-value[1]	Decision	Corrected p-value	Decision
$k4 > k1$	0.002	Reject	0.002	Reject
$k4 > k2$	0.002	Reject	1.000	Not Reject
$k4 > k3$	1.000	Not Reject	0.821	Not Reject
$k3 > k1$	0.006	Reject	0.086	Not Reject
$k3 > k2$	0.082	Not Reject	1.000	Not Reject
$k2 > k1$	0.505	Not Reject	0.068	Not Reject

Based on the results in Table 6.4, we reject the null hypothesis on the small dataset for $k_4 \leq k_1$, indicating that using 4 related subsets leads to a significantly higher mean F1 score than using only 1 subset. For the full dataset, we reject the null hypothesis for

[1]Using Bonferroni correction.

$k_4 \leq k_1$, $k_4 \leq k_2$, and $k_3 \leq k_1$, suggesting that using 3 or 4 related subsets leads to significantly better performance than using only 1 or 2 subsets.

These findings empirically demonstrate that on the full dataset, increasing the number of related subsets by 2 significantly improves the mean F1 score, and on the small dataset, increasing the number of related subsets by 3 leads to a similar improvement.

Based on the results presented in Table 6.3 of the validation set performance and the statistical analysis, we select $k = 4$ as the optimal number of related subsets for both the small and large datasets. This choice is supported by the fact that using $k = 4$ leads to statistically significant improvements in performance compared to lower values of k in most cases. While the improvement in F1 scores for $k = 4$ compared to $k = 3$ is not statistically significant, it consistently provides the best or near-best performance across various evaluation metrics and dataset sizes.

6.5 Impact of Classifier Head Configuration

In our initial model architecture Figure 4.4, we employed individual classifier heads for each of the $k + 1$ [CLS] tokens, resulting in a total of $69888 \times k + 1$ parameters. While this approach allows for potentially fine-grained predictions for each column and subset, it also increases the model's complexity and memory footprint.

To investigate whether this complexity is necessary, we explore an alternative, Figure 4.5 configuration with only two classifier heads: one for the target column's [CLS] token and one for all the [CLS] tokens from the related subsets. This configuration significantly reduces the number of parameters in the classifier to 69888×2.

We conducted experiments using both the original configuration (multiple classifier heads) and the simplified configuration (two classifier heads) with $k = 4$. Table 6.5 presents the F1 scores achieved by both configurations on the full and small datasets.

Table 6.5: Comparison of classifier head configuration.

Dataset Size	Strategy	Validation F1	Test F1						
		Micro	Micro	Macro	Weighted	MV	CC	FH	RC
Full	Multiple Heads	86.08	82.53	**82.72**	82.25	**83.94**	75.92	93.05	83.63
Full	Two Heads	**86.22**	**82.86**	82.69	**82.44**	83.90	**76.45**	**93.21**	**84.02**
Small	Multiple Heads	83.25	79.01	78.04	78.24	80.59	**73.20**	91.44	79.65
Small	Two Heads	**83.56**	**79.13**	**78.65**	**78.31**	**81.27**	72.60	**91.92**	**79.79**

The experiments comparing the multiple classifier heads to the simplified two-head configuration in our CTA model demonstrated that both configurations yield similar performance on both the full and small datasets. While the multiple-head configuration marginally outperformed the two-head configuration in terms of the macro-F1 score in some instances, the two-head configuration generally achieved superior results across other metrics, particularly the micro-F1 score. This is likely due to reduced overfitting in the simpler two-head model, as it may be less prone to capturing noise in the training data.

The model achieves a weighted-F1 score of 82.44, indicating good performance on most classes. The macro-F1 score is slightly higher at 82.69, suggesting balanced performance across all classes, regardless of their frequency. The lower weighted-F1 score

compared to the macro-F1 and micro-F1 scores suggests some class imbalance, with the model performing slightly better on more frequent classes. However, the performance difference is marginal.

These findings suggest that the simplified two-head configuration is a viable alternative, offering comparable accuracy with the added benefit of reduced model complexity and memory footprint. This streamlined architecture can be particularly advantageous in scenarios where computational resources are limited or efficiency is a priority.

6.6 Results on Test Set

To assess the effectiveness of our proposed CTA framework, we evaluate its performance on the SOTAB benchmark's test set, comparing it against state-of-the-art (SOTA) models. Table 6.6 presents the micro-F1 scores achieved by our model (using the two-head configuration with $k = 4$ related subsets), Random Forest (RF), TURL (Deng et al. (2022)), and Doduo (Suhara et al. (2022)) on various test subsets.

Table 6.6: SOTAB test set results.

Test Set	**Large Training Set**				**Small Training Set**			
	RF	**TURL**	**DODUO**	**OURS**	**RF**	**TURL**	**DODUO**	**OURS**
Full	58.58	78.96	**84.82**	82.86	55.57	72.16	76.27	**79.13**
Missing Values	60.47	73.14	83.28	**83.90**	58.04	66.98	69.55	**81.27**
Format Heterogeneity	64.78	90.14	92.98	**93.21**	62.35	87.88	85.95	**91.92**
Corner Cases	55.15	73.59	**78.03**	76.45	51.97	68.19	**74.00**	72.60
Random Columns	58.72	81.93	**87.12**	84.02	55.53	74.11	**81.82**	79.79

Our model demonstrates competitive performance across all test sets, achieving the highest micro-F1 score in two out of five categories ("Missing Values," and "Format Heterogeneity") for both the large and small training sets. This highlights its effectiveness in handling a wide range of CTA challenges.

When trained on the large dataset, our model performs comparably to the SOTA Doduo model, even slightly exceeding it in some categories. However, with the small dataset, our model significantly outperforms all other models, showcasing its ability to learn effectively from limited data.

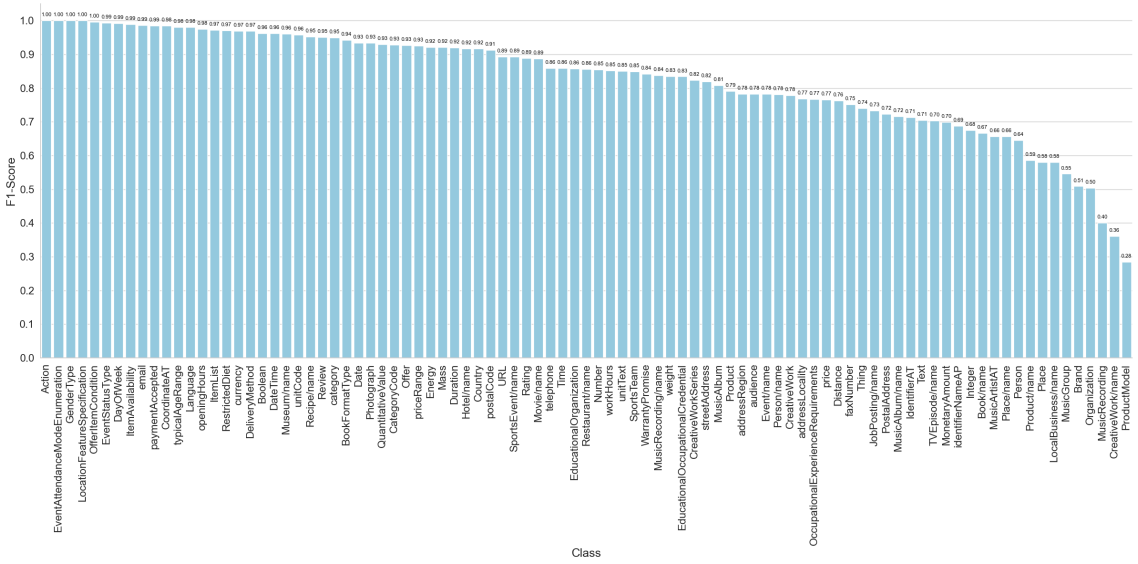
Our model achieves competitive results across most test sets, but its performance on the "Corner Cases" subset, which includes columns with ambiguous labels or atypical values, lags slightly behind Doduo. This suggests that our approach could be further refined to better handle these challenging column types.

Our proposed framework exhibits strong performance on the SOTAB benchmark, outperforming or matching state-of-the-art models in most test scenarios. The results validate the effectiveness of our approach, which uses pre-trained sentence transformers and Longformer's contextual modeling capabilities to accurately predict column types. This performance, especially with the small training set, underscores our framework's potential for real-world applications where labeled data might be scarce.

6.7 In-Depth Analysis of Per-Class Performance

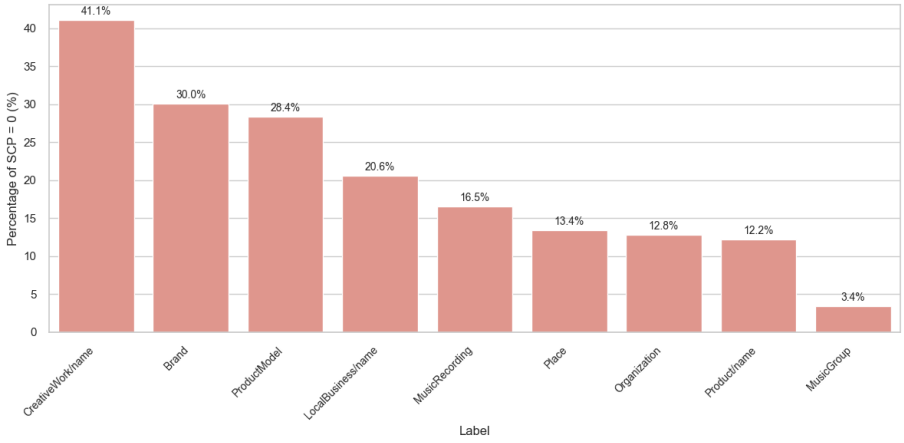
While the overall micro-F1 score (shown in Table 6.6) provides a useful summary of our model's effectiveness, it can mask nuanced variations in performance across different column types. To gain a deeper understanding of our model's capabilities and limitations, this section dives into a meticulous examination of its F1-score on a per-class basis across the test set. We will particularly focus on the cases where the model struggles and investigate potential reasons behind those challenges.

Figure 6.4: F1-class scores for on test set using full training dataset.



Analyzing the per-class F1-scores in Figure 6.4 for the full training dataset, we observed a notable drop from 0.64 to 0.59 among the 10 lowest-performing classes. This significant decrease suggests these column types pose particular challenges for our model. To investigate further, we analyze these classes in more detail.

Figure 6.5: Percentage of samples with zero similar columns.



We first examine the Similar Column Percentage (SCP), defined in Section 6.1, for these low-performing classes. Specifically, we focus on instances where SCP is zero, meaning the model found no similar columns in the related subsets during inference. Figure 6.5 illustrates the percentage of such instances in the training set for each of these labels.

For each class with SCP = 0 instances, we identified the top 5 labels that frequently lead to misclassification. We then calculated the average cosine similarity between the embeddings of the target class and these misleading classes. These results, presented in Figure 6.6, along with the percentage of misclassifications caused by each misleading class, provide insights into potential reasons for the model's struggles.

The analysis reveals several recurring patterns of misclassification:

- The "Product/name" label is often confused with the 'Text' label, likely due to columns containing generic text that is difficult to categorize. It also highlights confusion with other types of named entities (recipes, books, events, movies), indicating a need for features that better distinguish product names from other types of titles or names.

- The "Place" label often confuses specific address components and overlaps with "Organization" labels, which could be due to similarities in naming or descriptions in the data.

- The "LocalBusiness/name" label is frequently misclassified as 'Organization' and 'Place', indicating a challenge in distinguishing broader categories from specific sub-types.

- The "MusicGroup" label struggles with misclassification as 'Person/name', 'MusicRecording/name', 'MusicAlbum', and even other types of creative works like movies or series.

- The "Brand" label indicates difficulty in distinguishing brands from organizations and people, as well as generic text or locations, which might be mentioned in the context of a brand.

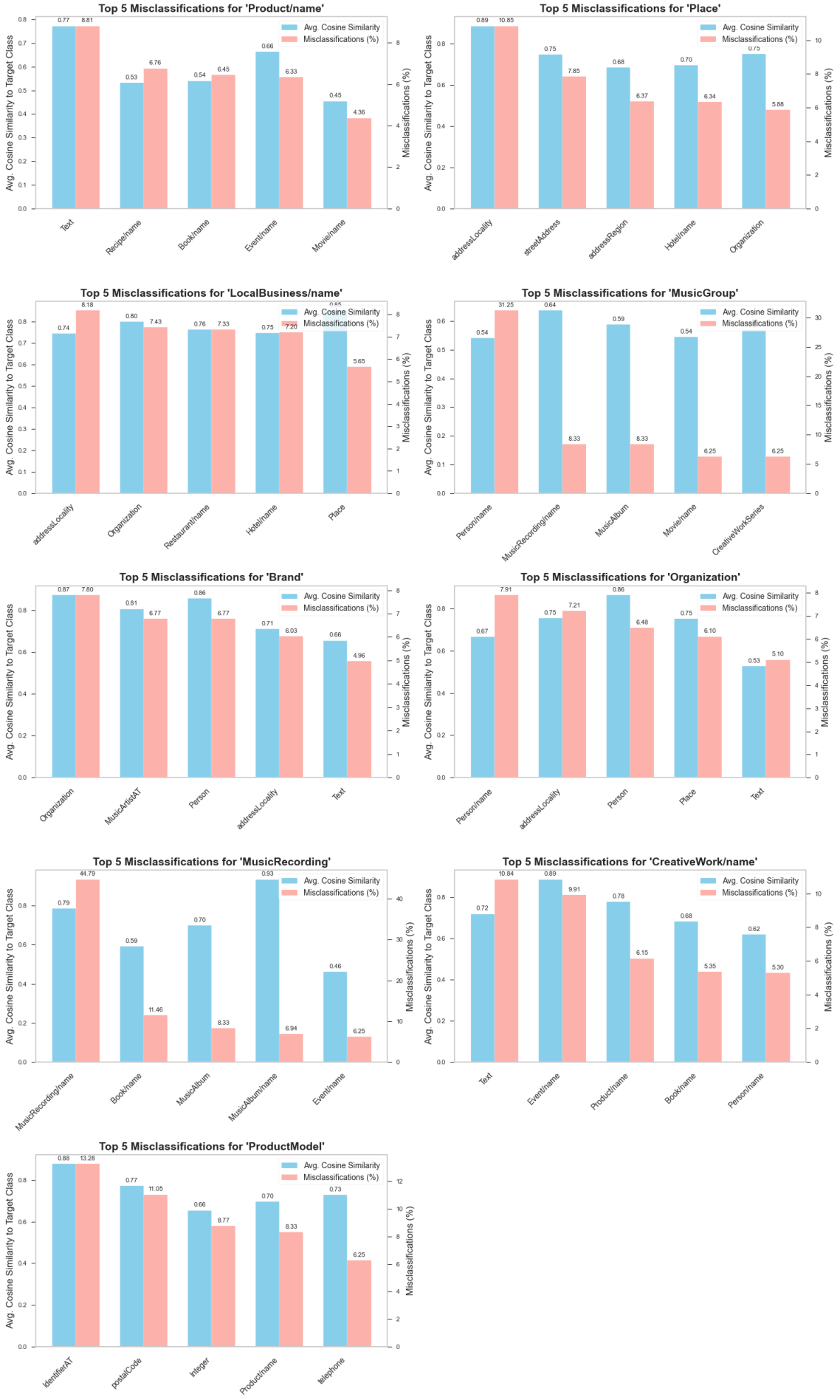
- Similar to "Brand", the "Organization" label reveals challenges in separating organizations from people and locations, as well as from generic text.

- The "MusicRecording" label reinforces the confusion within the creative works domain, particularly among different types of music-related entities. The "CreativeWork" label serves as a catch-all for various types of content, leading to confusion with more specific works (books, products) and events, as well as with people associated with them.

- The "ProductModel" label indicates a very specific type of product information that is often confused with identifiers, numerical values, product names, and even contact information, suggesting the model might rely too heavily on certain patterns (e.g., numbers, specific words) rather than understanding the broader context.

These results are likely due to semantic overlap, as evidenced by the high cosine similarity between the labels. For example, both ”Person” and ”Organization” columns might contain names, titles, or roles. Another contributing factor could be the lack of intra-table context; the meaning of a column might depend on the broader context of the table. For instance, a column named ”Location” could refer to a general place or the address of a local business.

Figure 6.6: Top 5 misclassifications and cosine similarity for each of the challenging labels.



Chapter 7

Conclusions

In this thesis, we presented a novel framework for CTA that effectively utilizes inter-table context and semantic similarity to enhance the accuracy and robustness of predictions. Through a series of experiments and analyses, we have addressed several key research questions:

- **RQ1 -** We demonstrated the effectiveness of sentence transformers in capturing semantic relationships between columns, enabling the identification of relevant contextual information across tables and significantly improving annotation accuracy.
- **RQ2 -** Our findings reveal a trade-off between token length and the number of related subsets (Figure 6.1). By optimizing these parameters, we achieved a balance between capturing rich context and maintaining computational efficiency. Additionally, we found that a simplified two-head classifier configuration (Table 6.5) can match the performance of a more complex model, reducing complexity without sacrificing accuracy.
- **RQ3 -** We confirmed that fine-tuning a pre-trained Longformer model significantly outperforms training from scratch, especially with limited labeled data (Table 6.2). This highlights the effectiveness of transfer learning for the CTA task.
- **RQ4 -** We identified specific column types, and columns considered hard to annotate particularly those in the "Corner Cases" subset, where our model struggles due to semantic overlap and limited context.
- **RQ5 -** Our framework demonstrated state-of-the-art performance on the SOTAB benchmark, outperforming or matching existing models in most test scenarios (Table 6.6). Remarkably, it excelled on smaller datasets, showcasing its ability to learn from limited data.

Our framework demonstrates its strong potential for real-world applications through its exceptional performance on the SOTAB benchmark. Particularly, on the "Value Format Heterogenity" and "Missing Values" sets, our model achieves a micro-F1 score of 93.21% and 83.90% for the large training set, outperforming the previous state-of-the-art model (DODUO). This translates to more accurate column type annotations, which can directly benefit various downstream tasks like data integration, semantic search, and knowledge base augmentation.

Importantly, our model excels when trained on smaller datasets and achieves a micro-F1 score of 79.13% on the full test set, thus beating the state-of-the-art model (DODUO) by 2.86%. Moreover, it demonstrated a remarkable ability to generalize effectively from limited labeled examples.

While our model demonstrates high accuracy on the majority of test cases, it's worth noting that its efficiency is also a significant advantage. The fine-tuning process (k=4 and two-head configuration) of the entire pre-trained model on the full dataset takes approximately 25 hours and 31 minutes on an NVIDIA TITAN V 32GB GPU, and training on the smaller dataset requires only 8 hours and 14 minutes. During inference, the model achieves a throughput of 6 annotations per second on a more affordable T4 16GB GPU, with a model size of 595MB.

By combining state-of-the-art performance with efficiency and adaptability to smaller datasets, our proposed framework opens up new possibilities for leveraging the vast amount of knowledge embedded in web tables.

7.1 Limitations and Future Work

Despite its promising performance, our framework has limitations. The model struggles with certain corner cases and semantically overlapping column types, indicating a need for further refinement. Additionally, the model's reliance on textual information might limit its applicability to datasets with predominantly numerical or mixed data types. Future work could focus on several directions:

- **Incorporating Numerical and Mixed Data:** Currently, our model primarily relies on textual information. Expanding the framework to effectively incorporate numerical and mixed-type data would enhance its applicability to a wider range of real-world scenarios.
- **Including Intra-Table Context:** While our approach successfully leverages inter-table context, integrating intra-table context (relationships within a single table) could further refine the model's understanding and improve its accuracy, particularly for semantically overlapping column types.
- **Leveraging External Knowledge:** Integrating external knowledge bases or ontologies could provide valuable information for disambiguating column types and validating predictions, leading to more accurate annotations.
- **Data Augmentation:** Exploring data augmentation techniques specifically tailored for column type annotation, such as rearranging related subsets or generating synthetic examples, could help address class imbalance and improve generalization.
- **Testing on Diverse Datasets:** To assess the framework's generalizability, it is essential to evaluate its performance on a wider range of datasets with varying characteristics, including different domains, schema structures, and label distributions.

By addressing these limitations and pursuing these directions of research, we can further advance the state-of-the-art in CTA and empower a broader range of applications in data understanding and integration.

References

Abdelmageed, N., Schindler, S., and König-Ries, B. (2021). Biodivtab: A table annotation benchmark based on biodiversity research data. In *SemTab@ISWC*.

Almeida, F. and Xexéo, G. (2019). Word embeddings: A survey. *ArXiv*, abs/1901.09069.

Arnold, A., Nallapati, R., and Cohen, W. W. (2007). A Comparative Study of Methods for Transductive Transfer Learning. In *Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007)*, pages 77–82, Omaha, NE. IEEE.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

Beltagy, I., Peters, M. E., and Cohan, A. (2020). Longformer: The long-document transformer. *ArXiv*, abs/2004.05150.

Bhagavatula, C. S., Noraset, T., and Downey, D. (2015). Tabel: Entity linking in web tables. In Arenas, M., Corcho, O., Simperl, E., Strohmaier, M., d'Aquin, M., Srinivas, K., Groth, P., Dumontier, M., Heflin, J., Thirunarayan, K., Thirunarayan, K., and Staab, S., editors, *The Semantic Web - ISWC 2015*, pages 425–441, Cham. Springer International Publishing.

Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., and Hellmann, S. (2009). Dbpedia - a crystallization point for the web of data. *Journal of Web Semantics*, 7(3):154–165. The Web of Data.

Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. In Burges, C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

Cassel, J. B. (2016). *Wolfram—Alpha: A Computational Knowledge "Search" Engine*, pages 267–299. Springer New York, New York, NY.

Chen, W., Wang, H., Chen, J., Zhang, Y., Wang, H., LI, S., Zhou, X., and Wang, W. Y. (2019). Tabfact: A large-scale dataset for table-based fact verification. *ArXiv*, abs/1909.02164.

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27.

Cutrona, V., Bianchi, F., Jiménez-Ruiz, E., and Palmonari, M. (2020). Tough tables: Carefully evaluating entity linking for tabular data. In *International Workshop on the Semantic Web*.

Cvetkov-Iliev, A., Allauzen, A., and Varoquaux, G. (2023). Relational Data Embeddings for Feature Enrichment with Background Information. *Machine Learning*, 112(2):687–720.

Deng, L., Zhang, S., and Balog, K. (2019). Table2vec: Neural word and entity embeddings for table population and retrieval. *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*.

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

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.

Eisenschlos, J., Gor, M., Müller, T., and Cohen, W. (2021). MATE: Multi-view attention for table transformer efficiency. In Moens, M.-F., Huang, X., Specia, L., and Yih, S. W.-t., editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7606–7619, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Gong, H., Sun, Y., Feng, X., Qin, B., Bi, W., Liu, X., and Liu, T. (2020). TableGPT: Few-shot table-to-text generation with table structure reconstruction and content matching. In Scott, D., Bel, N., and Zong, C., editors, *Proceedings of the 28th International Conference on Computational Linguistics*, pages 1978–1988, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Guha, R. V., Brickley, D., and Macbeth, S. (2016). Schema.org: evolution of structured data on the web. *Commun. ACM*, 59(2):44–51.

Harris, Z. (1954). Distributional structure. *Word*, 10(2-3):146–162.

Hearst, M., Dumais, S., Osuna, E., Platt, J., and Scholkopf, B. (1998). Support vector machines. *IEEE Intelligent Systems and their Applications*, 13(4):18–28.

Herzig, J., Müller, T., Krichene, S., and Eisenschlos, J. (2021). Open domain question answering over tables via dense retrieval. In Toutanova, K., Rumshisky, A., Zettlemoyer, L., Hakkani-Tur, D., Beltagy, I., Bethard, S., Cotterell, R., Chakraborty, T., and Zhou, Y., editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 512–519, Online. Association for Computational Linguistics.

Herzig, J., Nowak, P. K., Müller, T., Piccinno, F., and Eisenschlos, J. (2020). TaPas: Weakly supervised table parsing via pre-training. In Jurafsky, D., Chai, J., Schluter, N., and Tetreault, J., editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333, Online. Association for Computational Linguistics.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Hoffart, J., Suchanek, F., Berberich, K., Lewis-Kelham, E., de Melo, G., and Weikum, G. (2011). Yago2: Exploring and querying world knowledge in time, space, context, and many languages. pages 229–232.

Hogan, A., Blomqvist, E., Cochez, M., D'amato, C., Melo, G. D., Gutierrez, C., Kirrane, S., Gayo, J. E. L., Navigli, R., Neumaier, S., Ngomo, A.-C. N., Polleres, A., Rashid, S. M., Rula, A., Schmelzeisen, L., Sequeda, J., Staab, S., and Zimmermann, A. (2021). Knowledge graphs. *ACM Computing Surveys*, 54(4):1–37.

Hulsebos, M., Demiralp, , and Groth, P. (2023). Gittables: A large-scale corpus of relational tables. *Proceedings of the ACM on Management of Data*, 1(1):1–17.

Hulsebos, M., Hu, K., Bakker, M., Zgraggen, E., Satyanarayan, A., Kraska, T., Çağatay Demiralp, and Hidalgo, C. (2019). Sherlock: A deep learning approach to semantic data type detection.

Iida, H., Thai, D., Manjunatha, V., and Iyyer, M. (2021). Tabbie: Pretrained representations of tabular data.

Jiménez-Ruiz, E., Hassanzadeh, O., Efthymiou, V., Chen, J., and Srinivas, K. (2020). Semtab 2019: Resources to benchmark tabular data to knowledge graph matching systems. *The Semantic Web*, 12123:514 – 530.

Korini, K., Peeters, R., and Bizer, C. (2022). Sotab: The wdc schema.org table annotation benchmark. In *SemTab@ISWC*.

Liddy, E. (2001). Natural Language Processing. *School of Information Studies - Faculty Scholarship*.

Liu, J., Chabot, Y., Troncy, R., Huynh, V.-P., Labbé, T., and Monnin, P. (2023). From tabular data to knowledge graphs: A survey of semantic table interpretation tasks and methods. *Journal of Web Semantics*, 76:100761.

Liu, Q., Chen, B., Guo, J., Lin, Z., and Lou, J.-G. (2021). Tapex: Table pre-training via learning a neural sql executor. *ArXiv*, abs/2107.07653.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space.

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., and Taylor, J. (2019). Industry-scale knowledge graphs: lessons and challenges. *Commun. ACM*, 62(8):36–43.

Pan, F., Canim, M., Glass, M., Gliozzo, A., and Fox, P. (2021). CLTR: An end-to-end, transformer-based system for cell-level table retrieval and table question answering. In Ji, H., Park, J. C., and Xia, R., editors, *Proceedings of the 59th Annual Meeting of the*

Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations, pages 202–209, Online. Association for Computational Linguistics.

Pan, S. J. and Yang, Q. (2010). A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. ACL '02, page 311–318, USA. Association for Computational Linguistics.

Pennington, J., Socher, R., and Manning, C. (2014). GloVe: Global vectors for word representation. In Moschitti, A., Pang, B., and Daelemans, W., editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

Radford, A. and Narasimhan, K. (2018). Improving Language Understanding by Generative Pre-Training.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. In *Conference on Empirical Methods in Natural Language Processing*.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408. Place: US Publisher: American Psychological Association.

Rumelhart, D., Hinton, G., and Williams, R. (1988). Learning Internal Representations by Error Propagation. pages 399–421. Book Title: Readings in Cognitive Science.

Sennet, A. (2023). Ambiguity. In Zalta, E. N. and Nodelman, U., editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2023 edition.

Singh, R. and Bedathur, S. (2023). Embeddings for tabular data: A survey. *arXiv preprint arXiv:2302.11777*.

Singhal, A. (2012). Introducing the knowledge graph: things, not strings. 2020-11-13.

Suhara, Y., Li, J., Li, Y., Zhang, D., Demiralp, , Chen, C., and Tan, W.-C. (2022). Annotating columns with pre-trained language models. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD/PODS '22. ACM.

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

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. *arXiv.org*.

Tian, L., Zhou, X., Wu, Y.-P., Zhou, W.-T., Zhang, J.-H., and Zhang, T.-S. (2022). Knowledge graph and knowledge reasoning: A systematic review. *Journal of Electronic Science and Technology*, 20(2):100159.

Ugander, J., Karrer, B., Backstrom, L., and Marlow, C. (2011). The anatomy of the facebook social graph.

Vakulenko, S. and Savenkov, V. (2017). Tableqa: Question answering on tabular data. *ArXiv*, abs/1705.06504.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2023). Attention Is All You Need. arXiv:1706.03762 [cs].

Vesanto, J. (1999). SOM-based data visualization methods. *Intelligent Data Analysis*, 3(2):111–126.

Vrandečić, D. and Krötzsch, M. (2014). Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85.

Wang, D., Shiralkar, P., Lockard, C., Huang, B., Dong, X., and Jiang, M. (2021a). Tcn: Table convolutional network for web table interpretation. *Proceedings of the Web Conference 2021*.

Wang, F., Sun, K., Chen, M., Pujara, J., and Szekely, P. (2021b). Retrieving complex tables with multi-granular graph representation learning. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21. ACM.

Wang, Z., Zhang, J., Feng, J., and Chen, Z. (2014). Knowledge graph embedding by translating on hyperplanes. *Proceedings of the AAAI Conference on Artificial Intelligence*, 28(1).

Watkins, C. J. and Dayan, P. (1992). Technical Note: Q-Learning. *Machine Learning*, 8(3):279–292.

Yang, J., Gupta, A., Upadhyay, S., He, L., Goel, R., and Paul, S. (2022). TableFormer: Robust transformer modeling for table-text encoding. In Muresan, S., Nakov, P., and Villavicencio, A., editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 528–537, Dublin, Ireland. Association for Computational Linguistics.

Yin, P., Neubig, G., tau Yih, W., and Riedel, S. (2020). Tabert: Pretraining for joint understanding of textual and tabular data. *ArXiv*, abs/2005.08314.

Yu, T., Wu, C.-S., Lin, X. V., Wang, B., Tan, Y. C., Yang, X., Radev, D. R., Socher, R., and Xiong, C. (2020). Grappa: Grammar-augmented pre-training for table semantic parsing. *ArXiv*, abs/2009.13845.

Zhang, D., Suhara, Y., Li, J., Hulsebos, M., Çağatay Demiralp, and Tan, W.-C. (2020). Sato: Contextual semantic type detection in tables.

Zhang, N. and Jankowski, M. (2022). Hierarchical bert for medical document understanding. *ArXiv*, abs/2204.09600.

Zhang, S. and Balog, K. (2017). Entitables: Smart assistance for entity-focused tables. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17. ACM.

Appendix A

Algorithms

Algorithm 3 Create Column Embeddings

Input: Dataset D containing n target columns $D = \{C_1, C_2, \ldots, C_n\}$
Output: Dataset E containing n embeddings $E = \{E_1, E_2, \ldots, E_n\}$

//Step 1: Create list to store embeddings
$E \leftarrow$ empty list

//Step 2: Create column embeddings
for each target column C_i **in** D **do**
  $E_i \leftarrow$ SentenceTransformer(C_i)
  Append E_i to E
end for

return E

Algorithm 4 Finding Related Columns

Input: Target column embedding E^t, Dataset E containing n embeddings, Dataset D containing n target columns, number of related columns N
Output: Subsets of related columns H

//Step 1: Find the N^{th} most similar embeddings
$Sims \leftarrow$ SimilaritySearch(E, E^t, N)

// Step 2: Create $\frac{N}{4}$ subsets of related columns
$R \leftarrow [R_0, R_1, ..., R_{\frac{N}{4}-1}]$

//Step 3: Fill the subsets with 4 related columns
for each subset R_i **in** R **do**
  $indexes \leftarrow$ index$(Sims[i, i+4])$
  $R_i \leftarrow D[indexes]$
end for

return H

Algorithm 5 Serialization and Tokenization

Input: Target column C_j^t and subsets of related columns H
Output: Tokenized input sequence I_{token} local attention mask L_{attn} and global attention mask G_{attn}

// Step 1: Create input sequence
$I_{\text{token}} \leftarrow$ Empty sequence

// Step 2: Tokenize target column
$S(C_j^t) \leftarrow$ SerializeColumnValues(C_j^t)
$I_t \leftarrow$ LongformerTokenizer($S(C_j^t)$, 256)
$I_{\text{token}}.append([\text{CLS}], I_{R_i}, [\text{SEP}])$

// Step 3: Tokenize related columns
for each R_i **in** H **do**
  $I_{R_i} \leftarrow$ Empty sequence
  for each C_j **in** R_i **do**
    $S(C_j) \leftarrow$ SerializeColumnValues(C_j)
    Append LongformerTokenizer($S(C_j)$, 64) to I_{R_i}
  end for
  $I_{\text{token}}.append([\text{PAD}] \times 256), [\text{CLS}], I_{R_i}, [\text{SEP}])$
end for

// Step 4: Prepare Global and Local Attention Masks
$L_{\text{attn}}, G_{\text{attn}} \leftarrow$ LongformerTokenizer(I_{token})

return $I_{\text{token}}, L_{\text{attn}}, G_{\text{attn}}$

Algorithm 6 Serialization and Tokenization

Input: Target column C_j^t and subsets of related columns H
Output: Tokenized input sequence I_{token} local attention mask L_{attn} and global attention mask G_{attn}

// Step 1: Create input sequence
$I_{\text{token}} \leftarrow$ Empty sequence

// Step 2: Tokenize target column
$S(C_j^t) \leftarrow$ SerializeColumnValues(C_j^t)
$I_t \leftarrow$ LongformerTokenizer($S(C_j^t)$, 256)
$I_{\text{token}}.append([\text{CLS}], I_{R_i}, [\text{SEP}])$

// Step 3: Tokenize related columns
for each R_i **in** H **do**
  $I_{R_i} \leftarrow$ Empty sequence
  for each C_j **in** R_i **do**
    $S(C_j) \leftarrow$ SerializeColumnValues(C_j)
    Append LongformerTokenizer($S(C_j)$, 64) to I_{R_i}
  end for
  $I_{\text{token}}.append([\text{PAD}] \times 256), [\text{CLS}], I_{R_i}, [\text{SEP}])$
end for

// Step 4: Prepare Global and Local Attention Masks
$L_{\text{attn}}, G_{\text{attn}} \leftarrow$ LongformerTokenizer(I_{token})

return $I_{\text{token}}, L_{\text{attn}}, G_{\text{attn}}$

Appendix B

Loss Curves

Figure B.1: Losses for different configurations using large training set.

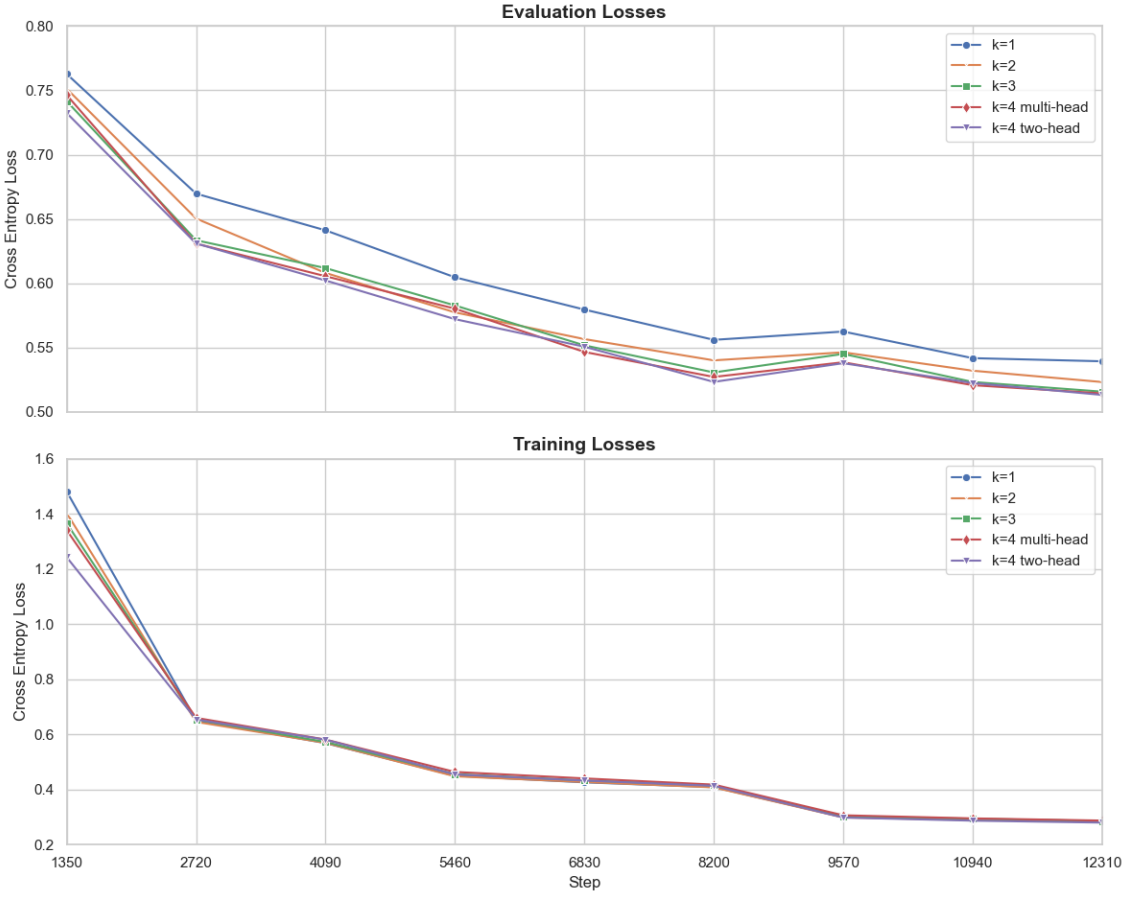
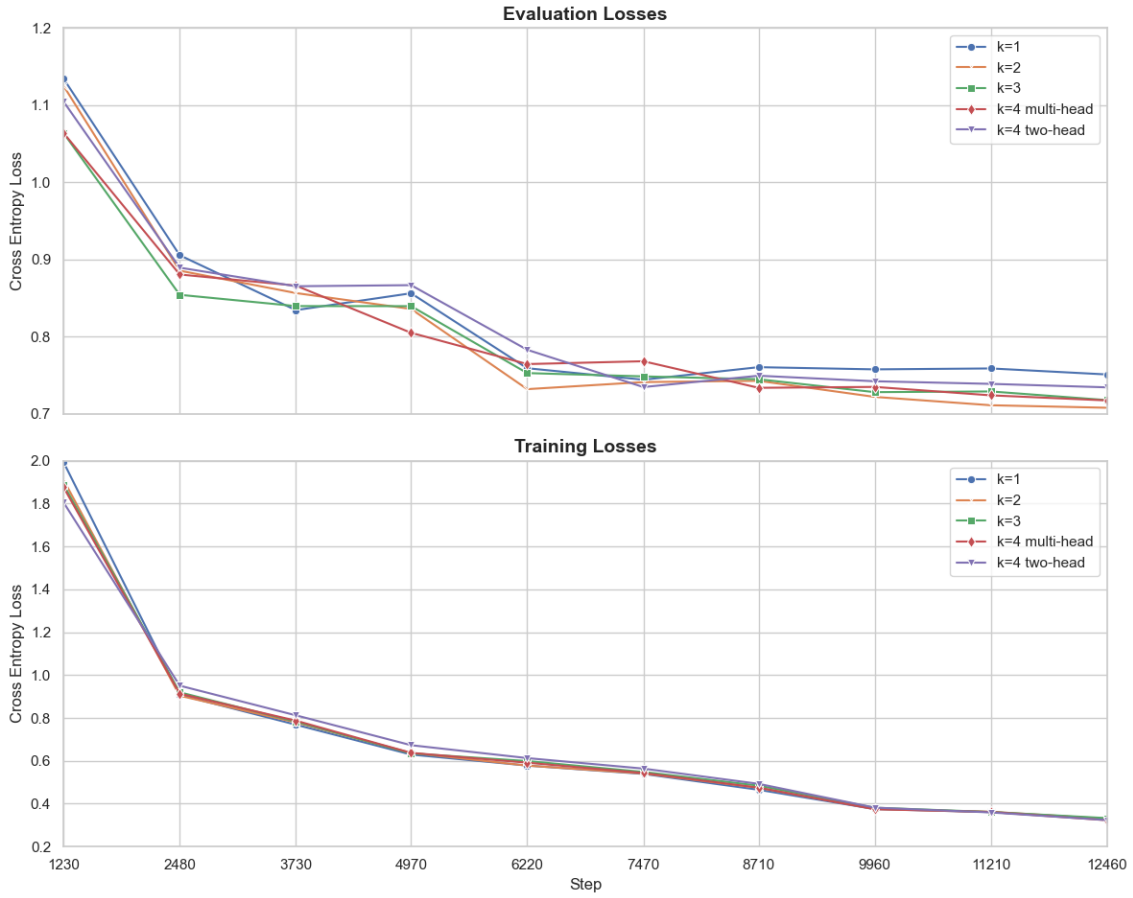


Figure B.2: Losses for different configurations using small training set.



NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa