Examining Data Integration with Schema Changes Based on Cell-level Mapping Using Deep Learning Models

by

Zijie Wang

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2021 by the
Graduate Supervisory Committee:

Jia Zou, Chair
Chitta Baral
Kasim Selcuk Candan

ARIZONA STATE UNIVERSITY

December 2021

ABSTRACT

Artificial Intelligence, as the hottest research topic nowadays, is mostly driven by data. There is no doubt that data is the king in the age of AI. However, natural high-quality data is precious and rare. In order to obtain enough and eligible data to support AI tasks, data processing is always required. To be even worse, the data preprocessing tasks are often dull and heavy, which require huge human labors to deal with. Statistics show 70% - 80% of the data scientists' time is spent on data integration process. Among various reasons, schema changes that commonly exist in the data warehouse are one significant obstacle that impedes the automation of the end-to-end data integration process. Traditional data integration applications rely on data processing operators such as join, union, aggregation and so on. Those operations are fragile and can be easily interrupted by schema changes. Whenever schema changes happen, the data integration applications will require human labors to solve the interruptions and downtime. The industries as well as the data scientists need a new mechanism to handle the schema changes in data integration tasks. This work proposes a new direction of data integration applications based on deep learning models. The data integration problem is defined in the scenario of integrating tabular-format data with natural schema changes, using the cell-based data abstraction. In addition, data augmentation and adversarial learning are investigated to boost the model robustness to schema changes. The experiments are tested on two real-world data integration scenarios, and the results demonstrate the effectiveness of the proposed approach.

DEDICATION

I dedicated this thesis to my family, who always support me to pursue my life and dream.

ACKNOWLEDGMENTS

First and the most, I would like to express my thanks to my advisor Dr. Jia Zou, who gives me the opportunity to start my research, who gives me constant guidance in daily work, who support me to finish this project and my thesis dissertation.

I would like to appreciate my thesis committee members, Dr. Chitta Baral and Dr. Kasim Selcuk Candan, thanks for your precious time and advice. I took the Natural Language Process course from Dr. Baral, which taught me solid knowledge that helped me a lot in this project.

Thanks to my colleagues that worked together in CACTUS lab, with all the advice I gained and the help I got. In particular, thanks to Lixi Zhou for his huge contributions to this project. Without your collaboration, my research will be less shiny and valuable.

Thanks to all the Professors in ASU that lectured me, with all those valueless knowledge, that gives me endless help in my study and research.

Thanks to my friends who helped me in my daily life and work.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

Artificial Intelligence (AI) has been investigated by researchers for several years. In recent, successful AI researches on Machine Translation[1], facial recognition[2], autonomous driving[3], and so on have changed our daily life profoundly. However, on the downside of AI, with the development of machine learning, especially in the era of deep learning, the size of deep learning models, as well as the size of data required to support the model training process, is unprecedentedly inflated. For example, GPT-3 (Brown et al. 2020), the largest language model developed so far, is built with 175 billion parameters and has been trained on a dataset with the size of more than 45 TB. The nature of deep learning makes it a heavily data-dependent task. On the other hand, we were never been in a crisis of data shortage. In fact, large amounts of data are generated every single day, through every daily activity and transaction. Unfortunately, most of the data generated are unmanaged, noisy, and scattered, which could not provide reliable sources for deep learning models training. Before any implementation of AI tasks, data processing tasks, such as data discovery, data cleaning, data integration and so on, are always needed.

Most of the data processing tasks are maintained and monitored by human labors,

[1]http://nlpprogress.com/english/machine_translation.html

[2]https://ai.google/responsibilities/facial-recognition/

[3]https://waymo.com/

approaches like crowdsourcing[4] are utilized to gather labor sources and have been recognized widely in AI area. However, considering the thirst of data for AI tasks and the low efficiency of human labors, crowdsourcing can not be the final answer. An alternative solution that minimizes human interventions is urgently needed to accelerate the process.

1.1 Challenges in Data Integration

Today, data is generated and distributed from various sources, with all kinds of formats, sizes, and contents. If data scientists want to conduct researches using certain data, it is not easy work for them to find just one table, with exactly all the contents they needed, a combination of different sources is required to collect the eligible data. Data integration is then proposed, in the early 1980s (Heimbigner and McLeod 1985), to solve the problem. Most of the solutions for data integration aim to find a mapping between source and target entities so that the source data schema can be converted into a unified global schema. Figure 1 shows an overview of a typical data integration task. For relational data, data integration could be as simple as a join operation, which combines two tables that share similar keys, but with different column contents. However, data integration is not always an easy task, it was reported in 2018 that data scientists spent 80%-90% efforts in the data integration process (Abadi et al. 2020) (Stonebraker, Ilyas, et al. 2018). The frequent changes and updates of data schema impact the data integration pipeline and cause system downtimes, and is always the main pain point that requires tremendous human resources involved in data integration. Schema changes are often caused by software

[4]https://www.mturk.com/

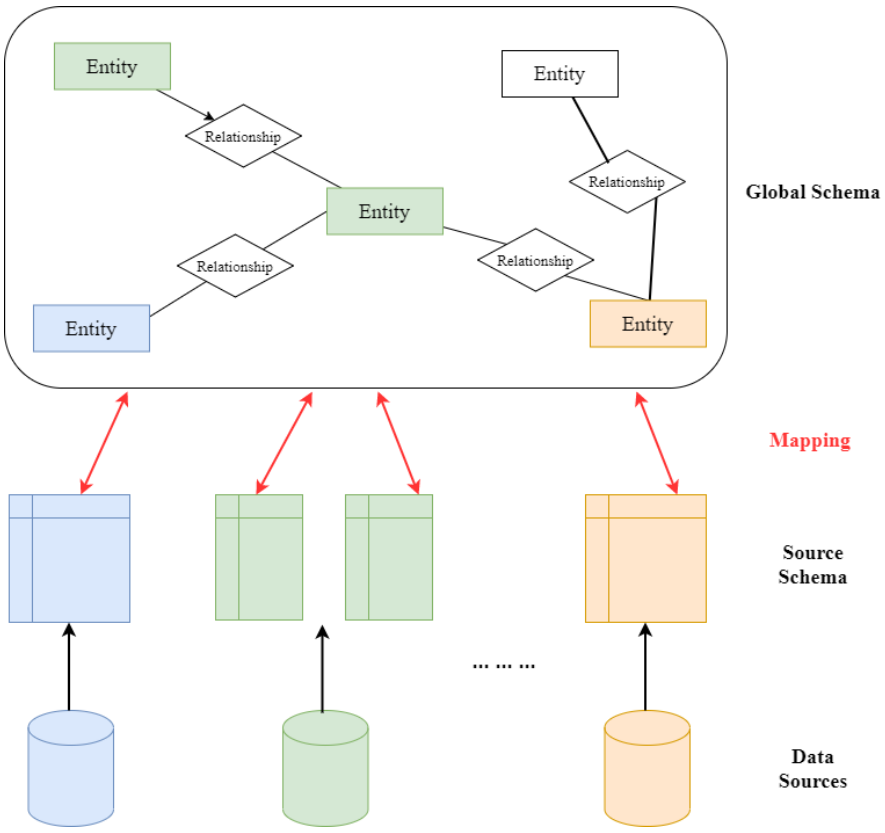


Figure 1. An overview of a typical data integration task

evolution that is pervasive and persistent in agile development (Howard 2011), or the diversity in data formats due to the lack of standards (Campbell et al. 2016).

The research topic of handling schema change for data managed in relational databases, NoSQL databases, and multi-model databases is well-studied. The fundamental idea of them is to capture the semantic mappings between the old and the new schemas so that the legacy queries can be transformed and/or legacy data can be migrated to work with the new schemas. There are two general approaches to capture the semantics mappings: (1) To search for the queries that can transform the old schema to the new schema (An et al. 2007) (Fagin et al. 2009) (Shen et al. 2014). (2) To ask the database administrators (DBAs) or application developers to use a domain-

specific language (DSL) to describe the schema transformation process (Bernstein and Melnik 2007) (C. Curino et al. 2013) (Moon et al. 2009) (Scherzinger, Klettke, and Störl 2013). However, these approaches are not applicable to open data, including publicly available CSV, JSON, HTML, and unstructured text files, as well as the potential schema changes that will happen in the future, since neither the history of schema changes nor the future of schema changes for these data can be recorded or predicted. It is an urgent need to handle such types of schema changes that minimize application interruptions and human interventions. Otherwise, with the rapid increase demands of data in the era of Big Data and Artificial Intelligence, it is unavoidable to waste a huge amount of time and human resources in manually handling the data integration application downtimes incurred by schema changes.

In the next section, we will illustrate a motivating example to further explain the application scenario of this work.

1.2 Motivating Example: COVID-19 Data Analysis

We start with a simple AI task. Assume a group of researchers plans to conduct a coronavirus outbreak analysis task. In order to avoid biases and errors, various factors need to be taken into consideration, like the number of COVID-19 confirmed case, recovered case, the change rate of population movement, and so on. However, one single data source only covers a part of the data needed, for example, COVID-19 data is maintained by medical organizations like Centers for Disease Control and Prevention (CDC), while the population mobility data is only provided by technology companies like Google. An integration of two or more data sources is required to get the target dataset.

Just as we mentioned before, Johns Hopkins University (JHU) is one of the organizations that maintain the worldwide coronavirus cases, the data is updated on a daily basis as a set of CSV files in their CSSE data repository[5]. During their daily updates, the schema changes, which could be the main bottleneck of the data integration task we proposed, happened frequently. We discovered several types of schema changes exist in the COVID-19 data repository. The changes include attribute name changes, attribute addiction and removal, attribute type changes, key changes. To handle one version of the schema change, data scientists have to hand-write code to process these data. However, the hand-write code can be easily broken by another version of schema change. This requires human efforts not only to write the processing code, but also to fix those issues. Even a separate data repository is maintained by data scientists to clean the original COVID-19 data into unified format [6]. In this work, we take this real-world scenario as one of our test cases, to better illustrate and validate our proposed approach. We will discuss more details in the following chapters.

1.3 Uninterruptible Integration of Schema Changed Data.

We use an example to illustrate how fragile the data integration pipeline is when schema change happens. Figure 2 shows a data integration task based on simple Python code. It utilizes APIs from Pandas DataFrame[7] to join two CSV files into

[5]https://github.com/CSSEGISandData/COVID-19

[6]https://github.com/Lucas-Czarnecki/COVID-19-CLEANED-JHUCSSE

[7]https://pandas.pydata.org/docs/reference/frame.html

Figure 2. Data integration on two CSV files using Python code

Note: Table Ratings_V2 is a schema-changed version of table Ratings.

one, simply based on their key. However, once the schema of one CSV file changes, such as the attribute name updates from **tconst** to **titleId**, the Python script will no longer work and requires human effort to debug and fix it. Each time such schema changes happened, extra labor will be needed.

In the past few years, Deep Learning (DL) (LeCun, Bengio, and Hinton 2015) has become the most popular topic in machine learning and artificial intelligence, and has deeply impacted a lot of research areas, such as computer vision, speech recognition, natural language processing, and so on. In recent years, many works

of database systems and applications investigate the utilization of deep learning to facilitate parameter tuning (Li et al. 2019) (Van Aken et al. 2017) (Zhang et al. 2019), indexing (Ding et al. 2020) (Kraska et al. 2018), partitioning (Zou et al. 2020), query optimization (Krishnan et al. 2018), and so on. While deep learning cannot guarantee 100% correctness, in the context of data integration tasks, errors are tolerable as long as most of the data is correct. This offers an opportunity for applying deep learning to data integration tasks. Our central hypothesis is: if we train neural network models to simulate and replace the programmer's hand-coded data integration application, the neural network models can be more robust and adaptable to schema changes than the fixed code, thus an uninterruptible integration pipeline will always work, no matter when and how often the schema changes happened.

In this work, we discuss and investigate the utilization of deep learning models to solve the data integration tasks, we abstract the data using a cell-level representation, we illustrate that our proposed approach works well in such data abstractions.

1.4 Chapter Organization

The organizations of this paper are summarized as follows. Chapter 2 discusses the previous works related to our problems, background knowledge utilized in this work will be explained. In Chapter 3, we investigate the stage of problem formulation, as well as the solution of the problem. In Chapter 4, we illustrate and explain the main design of our solution. In Chapter 5, we cover the model training process, the results are given and discussed. At last, Chapter 6 gives the conclusion and proposes the future works.

Chapter 2

RELATED WORKS AND BACKGROUND

In this chapter, we will first discuss the related works that exist in the database area. Background knowledge of deep learning models utilized in this work will also be explained briefly.

2.1 Related Works in Database

2.1.1 Schema Evolutions

Schema evolution in relational database, XML, JSON and ontology has been an active research area for a long time (Doan and Halevy 2005) (Rahm and Bernstein 2006). One major approach is through model (schema) management (Bernstein 2003) (Bernstein and Rahm 2000) and to automatically generate executable mapping between the old and evolved schema (Velegrakis, Miller, and Popa 2004) (Yu and Popa 2005). While this approach greatly expands the theoretical foundation of relational schema evolution, it requires application maintenance and may cause undesirable system downtimes (Curino, Moon, and Zaniolo 2008). To address the problem, Prism (Curino, Moon, and Zaniolo 2008) is proposed to automate the end-to-end schema modification process by providing DBAs a schema modification language (SMO) and automatically rewriting users' legacy queries. However, Prism requires data migration to the latest schema for each schema evolution, which may not be practical for today's Big Data era. Other techniques include versioning (Moon et

al. 2008) (Sheng 2019), which avoids the data migration overhead, but incurs version management burden and significantly slows down query performance. There are also abundant works discussing the schema evolution problem in NoSQL databases, Polystore or multi-model databases (Hillenbrand et al. 2019) (Holubová, Klettke, and Störl 2019) (Störl, Klettke, and Scherzinger 2020).

2.1.2 Data Discovery

Data discovery is to find related tables in a data lake. Aurum (Fernandez, Abedjan, et al. 2018) is an automatic data discovery system that proposes to build enterprise knowledge graph (EKG) to solve real-world business data integration problems. In EKG, a node represents a set of attributes/columns, and an edge connects two similar nodes. In addition, a hyperedge connects any number of nodes that are hierarchically related. They propose a two-step approach to build EKG using LSH-based and TF-IDF-based signatures. They also provide a data discovery query language SRQL so that users can efficiently query the relationships among datasets. Aurum is mainly targeting at enterprise data integration. In recent, numerous works are proposed to address open data discovery problems, including automatically discovering table unionability (Nargesian et al. 2018) and joinability (Zhu et al. 2019) (Zhu et al. 2016), based on LSH and similarity measures. Nargesian and et al. (Nargesian et al. 2020) propose a Markov approach to optimize the navigation organization as a DAG for a data lake so that the probability of finding a table by any of attributes can be maximized. In the DAG, each node of navigation DAG represents a subset of the attributes in the data lake, and an edge represents a navigation transition. All of these works provide helpful insights from an algorithmatic perspective and system

perspective for general data discovery problems. Particularly, Fernandez and et al. (Fernandez, Mansour, et al. 2018) propose a semantic matcher based on word embeddings to discover semantic links in the EKG.

2.1.3 Data Cleaning

The problem of data cleaning usually comes with two phases, identify the dirty data, and repair it. Chu and et al. (Chu et al. 2016) summarize and compare two ways that were widely used in data cleaning, the rule-based approaches that rely on integrity restriction (IR), and the statistical methods which take advantage of machine learning. ED2 (Neutatz, Mahdavi, and Abedjan 2019) utilize a two-stage active learning that can automatically learn the correct labels of samples, when lack of enough information, they proposed that their approach can reduce the dependence of user labeled samples. HoloClean (Rekatsinas et al. 2017) proposed to unify various data signals and information such as integrity restriction, data duplication and quantitative statistics, to construct a factor graph, then the factor graph is used to predict the right value of variables that needs to be fixed.

2.1.4 Schema and Entity Matching

Traditionally, to solve the data integration problem for data science applications, once related datasets are discovered, the programmer will either manually design queries to integrate these datasets, or leverage a schema matching tool to automatically discover queries to perform the data integration. There are numerous prior-arts in schema matching (Gottlob and Senellart 2010) (Kimmig et al. 2018) (Miller, Haas, and

Hernández 2000) (Alexe et al. 2011), which mainly match schemas based on metadata (e.g., attribute name) and/or instances. Entity matching (EM) (Christen), which is to identify data instances that refer to the same real-world entity, is also related. Some EM works also employ a deep learning-based approach (Ebraheem et al. 2017) (Kasai et al. 2019) (Konda 2018) (Mao et al. 2017) (Thirumuruganathan et al. 2018) (Zhao and He 2019). Mudgal and et al. (Mudgal et al. 2018) evaluate and compare the performance of different deep learning models applied to EM with three types of data: structured data, textual data, and dirty data (with missing value, inconsistent attributes and/or miss-placed values). They find that deep learning doesn't outperform existing EM solutions on structured data, but it outperforms them on textual and dirty data. In addition, to apply schema matching to heterogeneous data sources, it is important to discover schemas from semi-structured or non-structured data. Wang and et al. (Wang et al. 2015) proposed a schema discovery mechanism for JSON data, among other related works (DiScala and Abadi 2016) (Mior et al. 2017).

2.2 Deep Learning in Natural Language Process

In recent several years, Natural Language Processing (NLP) has benefited several major advancements from the fast development of deep learning models. Sequence models like Recurrent Neural Network (RNN) (Elman 1990) are enhanced by novel mechanisms like Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber 1997) and Gated Recurrent Unit (GRU) (Cho et al. 2014). Attention mechanism was then applied to sequence model and proved to be successful (Bahdanau, Cho, and Bengio 2014) (Luong, Pham, and Manning 2015). In the very recent, the newly proposed Transformer (Vaswani et al. 2017), a model which takes advantage

of the attention mechanism while discards the complicated and inefficient sequence architecture, has already changed the NLP world. The Transformer model and its variations achieve state-of-the-art (SOTA) performance on various NLP tasks, including Machine Translation (MT) (Edunov et al. 2018) (X. Liu et al. 2020), Question Answering (QA) (Radford et al. 2018), Natural Language Inference (NLI) (Y. Liu et al. 2019) and so on.

2.2.1 Word Embedding

Deep learning models take numerical vectors or matrices as input. In computer vision, images can be decomposed to pixels and then converted to numerical values, thus the whole images can be represented by matrices. However, there is no natural way to convert text data to numerical values. From the statistical perspective, approaches like Term Frequency Inverse Document Frequency (TF-IDF) (Ramos et al. 2003) tries to use the frequency of word existing in the corpus to represent each word, the higher frequency will be assigned with larger values. This approach is less effective since it loses extra information like the co-occurrence tendency among words, which is a common case in natural language. On the other hand, people investigate to model this problem from the learned perspective. We will cover several word embedding methods based on learned model in the following content.

2.2.1.1 Word2vec

Word2vec (Mikolov et al. 2013) is a learned-based approach that utilized a two-layer neural network to learn the representation of words. To be more specific, the

vector representation of one word is affected by its context words. One imperfection of Word2Vec is that it only considers the vector representation in word level, for example, one word with different forms (e.g., 'run' & 'running') is not considered as similar during its learning process.

2.2.1.2 fastText

fastText (Bojanowski et al. 2016) improves the method of Word2vec by considering the vector representation further in the character level. For example, the word **running** is treated as a group of separated characters (**run, unn, nni, nin, ing**), therefore its similarity with the word **run** can be discovered. On the other hand, the length of the characters can be set during the training process, which brings more adaptation for various text sources.

2.2.1.3 WordPiece

WordPiece (Wu et al. 2016) is a subword tokenization tool that is primarily used by Transformer models. In order to solve the problem with Out-Of-Vocabulary (OOV) words, WordPiece is designed to separate a single OOV word into several subwords or characters, and then trained with a deep LSTM model. During the training process, parallel training including data parallelism and model parallelism were applied to reduce the training time latency.

2.2.2 Basics of Bi-LSTM

Bidirectional Long Short-Term Memory (Bi-LSTM) is modified and improved from Recurrent Neural Network (RNN), it improves the training stability and performance by solving the vanishing gradients problem (Pascanu, Mikolov, and Bengio 2013). In its design, each hidden state in RNN that only does a linear combination of previous state and current state is replaced by a LSTM unit, while each LSTM unit is constructed by three different gates, input gate, forget gate and output gate. In order to collect the information from both sides of an input sentence, a backward LSTM layer is added after the forward LSTM layer. At last, the output of forward layer and backward layer will be concatenated to form the model output. Figure 3 shows the overview architecture of a typical Bi-LSTM model.

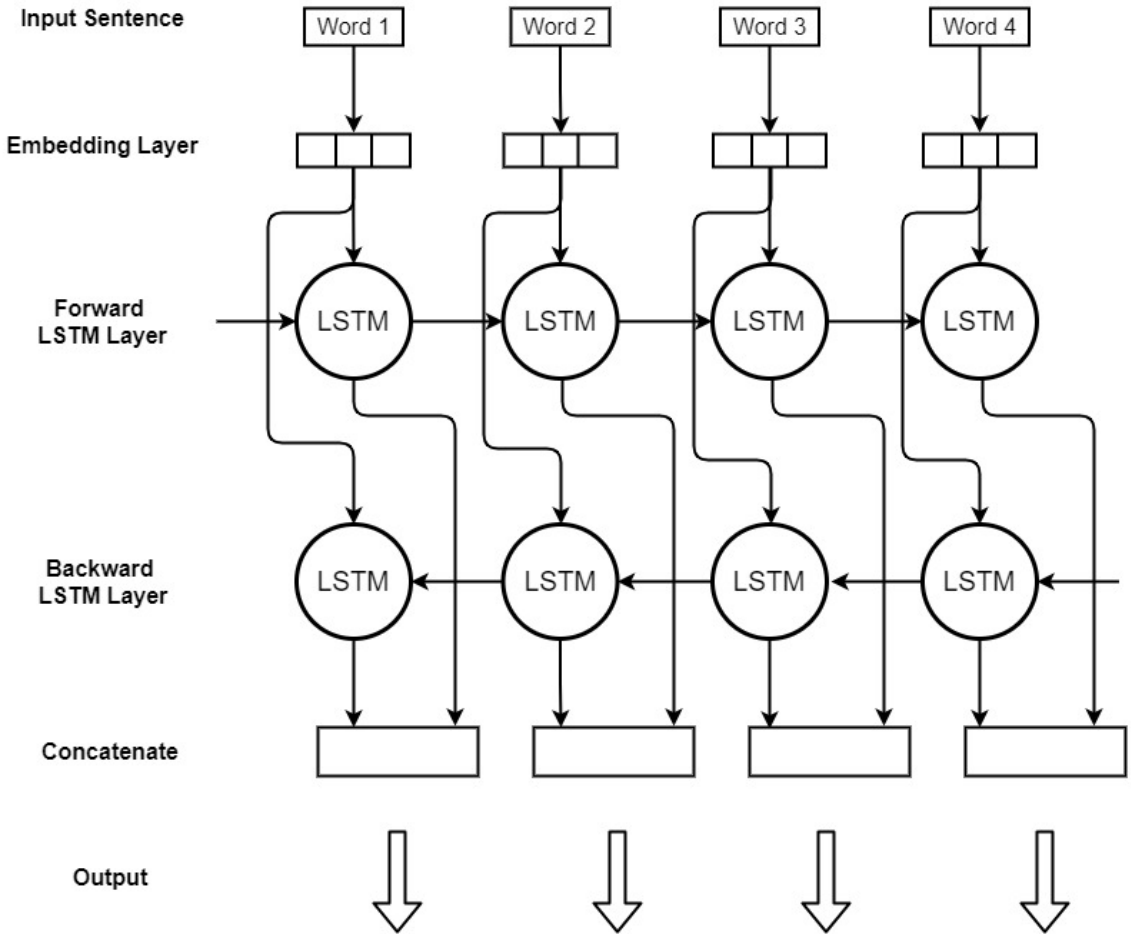


Figure 3. Typical architecture of a Bi-LSTM model

Note: The representation of input words (i.e., subwords, characters, etc.) may vary when using different approaches of word segmentation.

The LSTM architecture is mathematically defined as follows in (Goldberg 2016):

$$
\begin{aligned}
s_j = R_{LSTM}(s_{j-1}, x_j) =& [c_j; h_j] \\
c_j =& c_{j-1} \odot f + g \odot i \\
h_j =& \tanh(c_j) \odot o \\
i =& \sigma(x_j W^{xi} + h_{j-1} W^{hi}) \\
f =& \sigma(x_j W^{xf} + h_{j-1} W^{hf}) \\
o =& \sigma(x_j W^{xo} + h_{j-1} W^{ho}) \\
g =& \tanh(x_j W^{xg} + h_{j-1} W^{hg})
\end{aligned}
\tag{2.1}
$$

$$
y_j = O_{LSTM}(s_j) = h_j
$$

in which

$$
s_j \in \mathbb{R}^{2 \cdot d_h},\ \ x_i \in \mathbb{R}^{d_x},\ \ c_j, h_j, i, f, o, g \in \mathbb{R}^{d_h},\ \ W^{x\circ} \in \mathbb{R}^{d_x \times d_h},\ \ W^{h\circ} \in \mathbb{R}^{d_h \times d_h};
$$

While x_j represents the input vector in position j, i represents the input gate, f represents forget gate, o represents the output gate.

2.2.3 Architecture of Transformer

Unlike the sequence model, the Transformer model takes advantage of the Encoder-Decoder architecture as well as the attention mechanism. Figure 4 illustrates the architecture of Transformer model that was proposed originally (Vaswani et al. 2017). At first, a learned embedding layer is used to transform the input sentence to embedding vectors; positional embeddings are then added along with the word embeddings and

then pass through the encoder (decoder) blocks. Different from the self-attention used in encoder block, the masked self-attention is applied in decoder block to get the sequence only from the forward direction, since the ground truth of the latter words may affect the prediction results of former words.

The original Transformer model was only proposed to solve Machine Translation tasks, while researchers investigate to extend a pretrained version of the model to more scenarios. We will discuss it in the next subsection. In the next part, the mechanism of self-attention will be discussed.

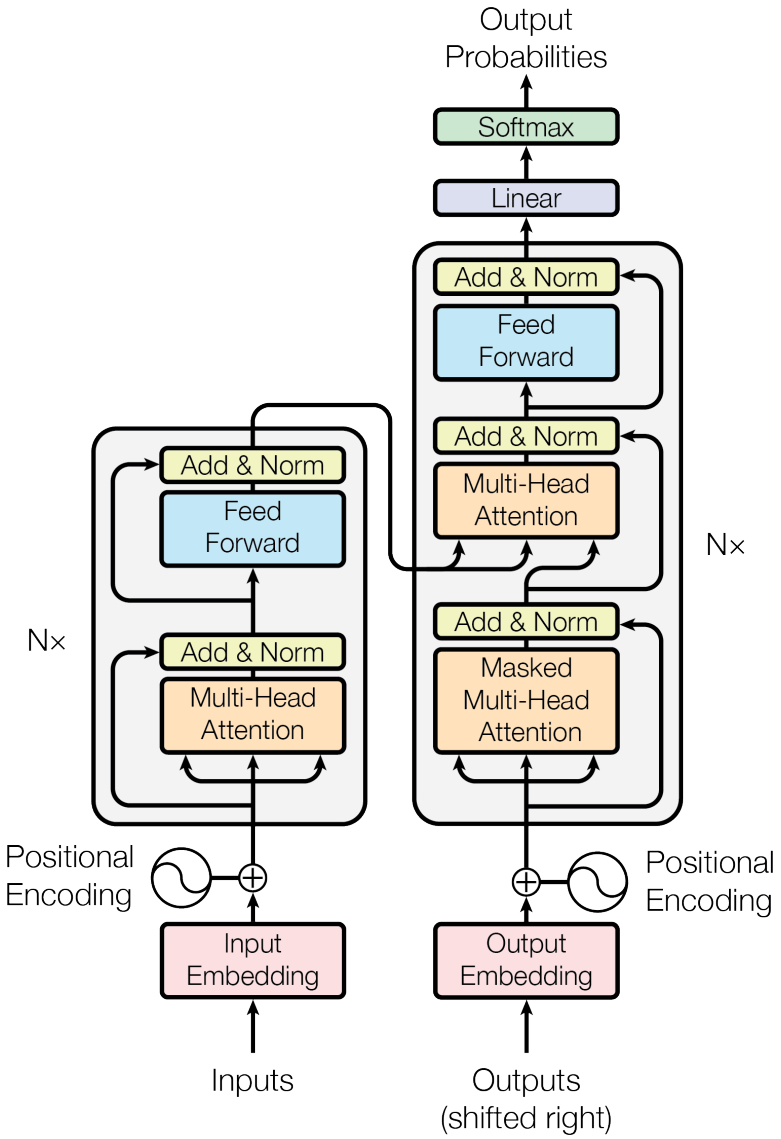


Figure 4. Transformer model architecture

Source: Vaswani et al. (2017)

2.2.3.1 Self-Attention

The self-attention mechanism, which is the key component of the Transformer model, is based on an assumption that, if the model takes more focus on a smaller but more important part that is useful for current prediction, instead of considering the entire input sequence, it can largely promote the effectiveness and correctness of the large and deep models. The self-attention takes three components as input, query Q, key K, and value V; The main idea is to map the query to a set of key-value pairs, the more likely the query and the key, the more weight the corresponding value will be assigned. The output is then produced by the weighted values. The equation for computing the output is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.2}$$

2.2.4 Architecture of BERT

Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al. 2018) is a successful extension of Transformer model. Unlike the early investigations that only trained and implemented the model on Machine Translation tasks, later works proposed to pretrain the transformer model on a huge but task-irrelevant dataset, and then fine-tuning the stored pretrained model on the target dataset (a.k.a. Transfer Learning), which is proved to be effective on various NLP tasks (Radford et al. 2018) (Howard and Ruder 2018). BERT, as its name, is formed only by the encoder block, and takes the input sequence from both forward and backward directions. To solve the same problem that met in the original Transformer model, instead of using the masked self-attention layer, BERT designs to randomly masked out several words

in input sequence, and then tries to predict the masked tokens during the pretraining process.

The implementation of BERT mainly includes two steps, an unsupervised pretraining process and a supervised fine-tuning process.

2.2.4.1 Pretraining

Before directly applying the BERT model to the target scenario, the barebone model needs to be pretrained on a huge volume of corpus to get initialized. Firstly, the input sentence is converted to tokens by using WordPiece tokenizer, such process is named tokenization. Before getting input into the model, 15% of the whole tokens are randomly chosen to be masked out, among them (1) 80% will be replaced by **[MASK]** token; (2) 10% will be replaced by a random token; (3) the rest of 10% will keep unchanged.

The pretraining process is designed to cover two objectives.

- **Masked Token Prediction** Predict the **[mask]** token based on its neighbor tokens.
- **Next Sentence Prediction** Some NLP tasks (e.g., Question Answering, Natural Language Inference) aim to predict the relationship between two sentences. The next sentence prediction is designed to help the model to learn a better understanding of sentence relationships.

2.2.4.2 Fine-tuning

A pretrained model works well, but only on the particular dataset, especially on those that have been seen during the training process. This is far away from satisfactory since once the dataset or target downstream tasks change, all the resources and time cost on the previous pretrained model are wasted. The fine-tuning process is designed to maximize the use of the pretrained models. Based on the feature of new input samples, the model tries to adjust its learned weight and thus has a better adaptation to the new dataset. A common way to implement the fine-tuning is simply adding a fully connected (FC) layer and softmax layer after the output of pretrained model. This design makes the fine-tuning process fast and low-cost. Ideally, for every different type of dataset or downstream task, we can fine-tuning a unique model, based on the same pretrained model. The fine-tuned BERT models obtain SOTA performance on various downstream tasks.

Chapter 3

PROBLEM FORMULATION AND STUDY

Data integration requests usually come from the researchers and users who were not familiar with the basic knowledge of database systems. We formulate our data integration problem as a real-world request of integration task required by non-experts. Users only have to provide the schema of their expected target dataset, and the candidate sources dataset they want to extract content from. Figure 5 shows the overview of our formulation of the data integration problem.

3.1 Problem Formulation

3.1.1 Preliminary Assumption

Firstly, even though the source datasets may have heterogeneous formats such as CSV, JSON, text, etc., it is a relatively easy work to parse those data to tabular format. We assume that in a typical data integration scenario that converts a set of source datasets into one target dataset, each of the source datasets will be preprocessed to a tabular table with clearly defined keys and attributes. On the other hand, the target dataset defined by each integration task must be tabular too. In this situation, each cell in the target dataset can be uniquely identified by its row identifier and attribute name.

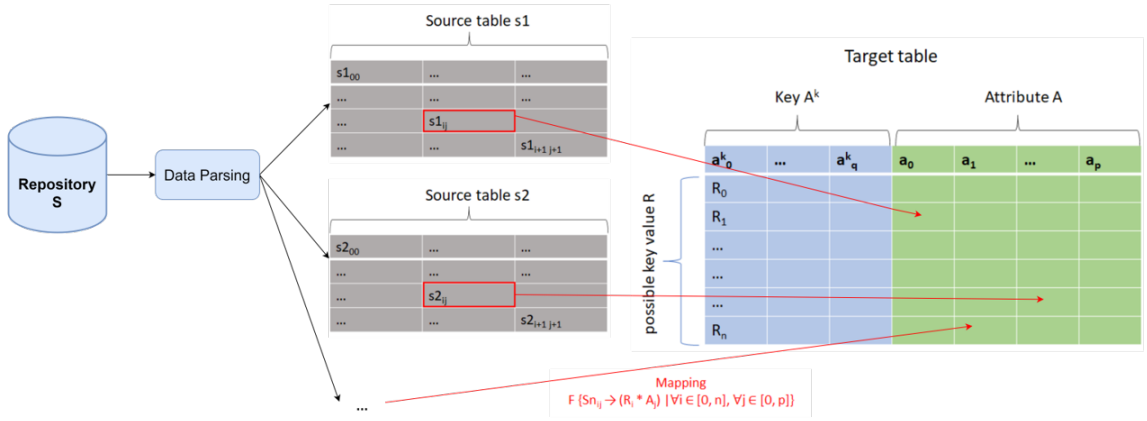


Figure 5. Data Integration: modeling the mapping from source repository to target table

3.1.2 Problem Statement

We address the problem of data integration with schema changes as follows: Given a fast-evolving data repository $S = \{Sn_{ij} \in Sn | \forall Sn \in S\}$, where Sn represents the n-th table that extracts from the repository through a data parsing process, with a clearly defined schema. For each data integration request, the user input specifies the expected target table schema, with number of p attributes $A = \{a_j\}(0 \leq j \leq p)$, as well as a length of q attributes array $A^k = [a_0^k, ..., a_q^k]$ that serves as key. The user further denotes a set of existing possible key values in the target table as $R = \{R_i\}(0 \leq i \leq n)$. Based on the aforementioned conditions, we aim to predict data sample Sn_{ij} (we further abstract the sample Sn_{ij} based on various data granularities) based on its key value Sn_i and attribute value Sn_j to its target position $[R_i, A_j]$ using the model $f\{Sn_{ij} \rightarrow (R_i * A_j) \ | \forall i \in [0, n], \forall j \in [0, p]\}$. Samples from source tables that $\{Sn_{ij} | i \notin [0, n] \vee j \notin [0, p]\}$ will be taken as discard.

3.2 Problem Study

3.2.1 Modeling Data Integration as Prediction Task

We start from the hypothesis proposed before: The deep learning models will be more robust to the schema changes than traditional rule-based integration systems, thus the data integration pipeline won't be easily interrupted by schema changes. We support our idea by starting with several discoveries.

First, the traditional data integration application actually tries to encode the mapping relationship between the entities in the source datasets and the entities in the target datasets, such non-linear relationship is usually represented through a set of data processing operators such as join (including 1-1 join and 1-many join), aggregate, filter, projection, union, and so on. On the other hand, modern deep learning models, especially supervised learning, also aim to learn the non-linear relationship between the input sample features and the assigned labels. It is natural thinking to replace the traditional operator-based data integration, using deep learning models. In this work, we will demonstrate that a broad class of data integration programs consisting of these operators can be formulated as a predictive problem and modeled using state-of-the-art neural network models.

Second, we find the data integration application code based on data processing operators will be easily interrupted by schema changes. However, the neural network models are used to handle the variety and can be further trained with robustness to handle most types of schema changes. The feature-label representation of a predictive problem is significantly more flexible than the fixed input formats expected by each data processing operator in a dataflow computation graph. For example, if several

related source datasets are denormalized into a wide dataset, it will interrupt a program that contains a join operation of the original source datasets. While it won't affect a neural network model, in which each training sample could represent a cell (or higher granularity like a multi-cell and a tuple), which is the smallest granularity of data elements that is parsable from a source dataset, such as a field in a CSV or tabular file, a leaf node in a JSON object, or a token in a free text file. The needed data processing operators become different, while the features (e.g., contexts) of each source cell remain the same after such denormalization. In addition, data augmentation, where various pre-defined schema changes can be generated and added to training data, and adversarial learning, where perturbations are added during the training process, could be utilized to make the model robust to schema changes.

3.2.2 Modeling Source Data with Various Data Granularity

No matter what initial format the source dataset is, we convert it to tabular tables based on the data parsing process, as we illustrated before. Then the tabular-format source dataset can be viewed as a set of samples, during the model training process. We observed potentially at least five candidate abstractions of the samples for the prediction problem formulation: dataset-based, object-based, supercell-based, attribute-based, and cell-based. Figure 6 illustrates an example of various levels of representations.

We find it almost impossible to collect sufficient training data represented at the dataset-level, thus we will not discuss the dataset-level representation in the rest of the paper.

For object-based abstraction, each training sample represents an object, and labels

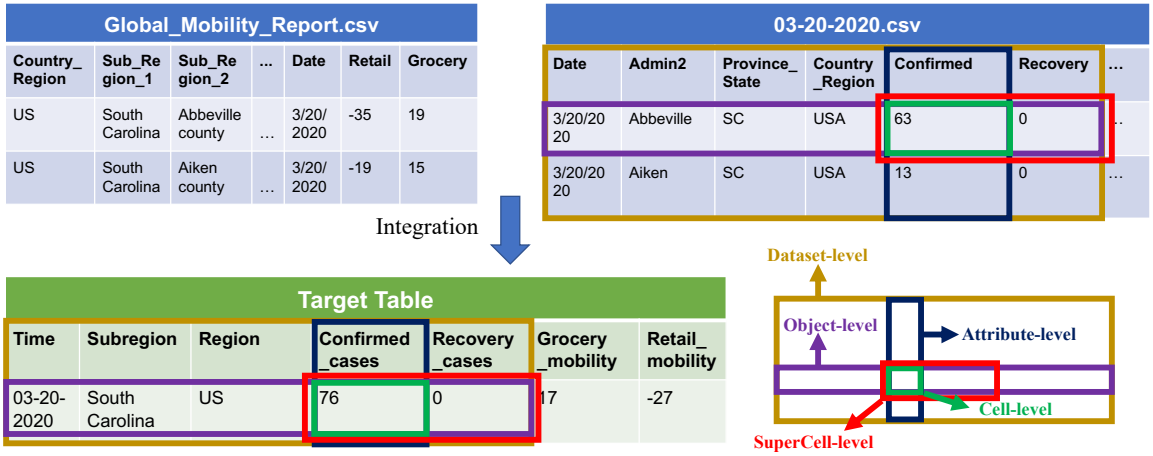


Figure 6. In this example, we define five level of abstractions: dataset-based, object-based, supercell-based, attribute-based, and cell-based.

specify how the object is mapped to the target table, e.g., which target rows the object is related to, how each attribute in the object is mapped to the attributes in the target table, and the aggregation function of each target attribute. The problem is that at this coarse granularity, too many labels need to be introduced, which will greatly reduce the prediction accuracy, as to our observation.

For attribute-based abstraction, each training sample represents an attribute in a source dataset. Some attributes exist in only a few objects, while some attributes are shared by most of the objects. Labels are used to indicate how a source attribute is mapped to a target attribute. Although a predictive problem in this abstraction can reveal the joinable or unionable attributes and can facilitate simple equi-join and union, and can easily perform attribute projection tasks. Unfortunately, it cannot express many other data processing logics such as filtering, aggregation, and join with complicated predicates, because these operations require information regarding individual objects, which are missing in this abstraction. In addition, if the task is simply to tell the mapping relationship between source and target attributes, using locality sensitive hashing to compute a signature for each attribute based on all of its

values will be more straightforward than a deep learning approach, as demonstrated in many data discovery researches (Miller 2018) (Zhu et al. 2019).

Given the shortcomings of the above abstractions, we decided to adopt a multi-level granularity-based abstraction. The first is the cell-based abstraction, where a cell is the value of an attribute in an object, and it is the smallest parsable unit of data. Each cell in the source datasets is described using features extracted from its context, including the attribute name of the cell, and the non-numerical attribute values in the object, such as the categorical values, and self-growing values. The prediction labels associated with each cell include three parts: the cell's key identifier in the target table, its column identifier in the target table, and the aggregation functions that will be applied to all cells mapped to the same position (e.g., sum, average, max, min, replace old values, discard new values).

Secondly, we chose another supercell-based abstraction, which sightly enlarged the granularity compare to cell-based abstraction. A supercell is a group of cells that always show up together in the source table and always be mapped together in the target table. For example, attribute **Confirmed** and **Recovery** that shown in Figure 7, are considered as a supercell in the data abstraction. The information needed to construct the supercell abstraction, for example, which attributes are more likely to show up together, how many attributes should be included in the supercell, is provided by each data integration request.

The coarser-grained of the representations, the fewer times of inferences are required, and the more efficient the prediction is. However, a coarser-grained representation also indicates that the prediction task is more complex and harder to train a model with acceptable accuracy, because a training sample will be larger and more complex than other fine-grained representations, and the mapping relationship to learn will

naturally become more complicated. We will flexibly leverage the trade-off between the time overhead and performance.

3.2.3 Handling Schema Changes

Schema changes in RDBMS (C. A. Curino et al. 2008) (Sjøberg 1993), XML/JSON (Moro, Malaika, and Lim 2007), NoSQL stores (Scherzinger and Sidortschuck 2020), ontology (Stojanovic 2004), and object-oriented databases (Banerjee et al. 1987), have been well characterized. We identify a number of schema changes that will interrupt user programs, such as dimension pivoting, attribute ordering change, key ordering change, removal of irrelevant attributes, removal of irrelevant datasets, denormalization, normalization, renaming of attributes, reformatting of cell values, key expansion, and so on. Our cell-based representation will not be affected by dimension pivoting, attribute ordering change, denormalization, and normalization of datasets, because the context for a cell remains the same in these cases. Therefore, a model trained with our proposed representation is by its nature robust to these types of schema changes. There remain four types of schema changes that will confuse our proposed model: (1) the renaming of attributes; (2) reformatting of cell values; (3) key expansion; (4) key ordering changes.

To address the problems, we inject corresponding perturbations to augment training data so that the training process is aware of and can be robust to these schema changes. The details of data augmentation will be discussed in Chapter 4.

Chapter 4

SYSTEM DESIGN: DATA INTEGRATION BASED ON DEEP LEARNING MODELS

In this chapter, we present and explain the details of our proposed method based on deep learning models. As shown in Figure 7, during the training stage, the existing data sources at earlier timestamp will be used as training dataset, with data augmentation applied. While during the inference stage, even if the data sources have been changed through many times of evolutions, the trained models are still robust enough to handle them.

4.1 Applicable Data Integration Scenario

We start from the data integration scenarios we mainly focus on, COVID-19 data and Machine Log data. These two are real-world datasets that have either been through frequent schema evolutions during the development of data repository, or have been covered with natural schema variations because of the various data sources. Based on that, we considered them as applicable integration scenarios to validate our proposed approach. We will explain each of these datasets in the following content.

4.1.1 COVID-19 Data Integration

As described in the motivating example, in this scenario, we try to integrate two independent data sources, the coronavirus data repository maintained by Johns

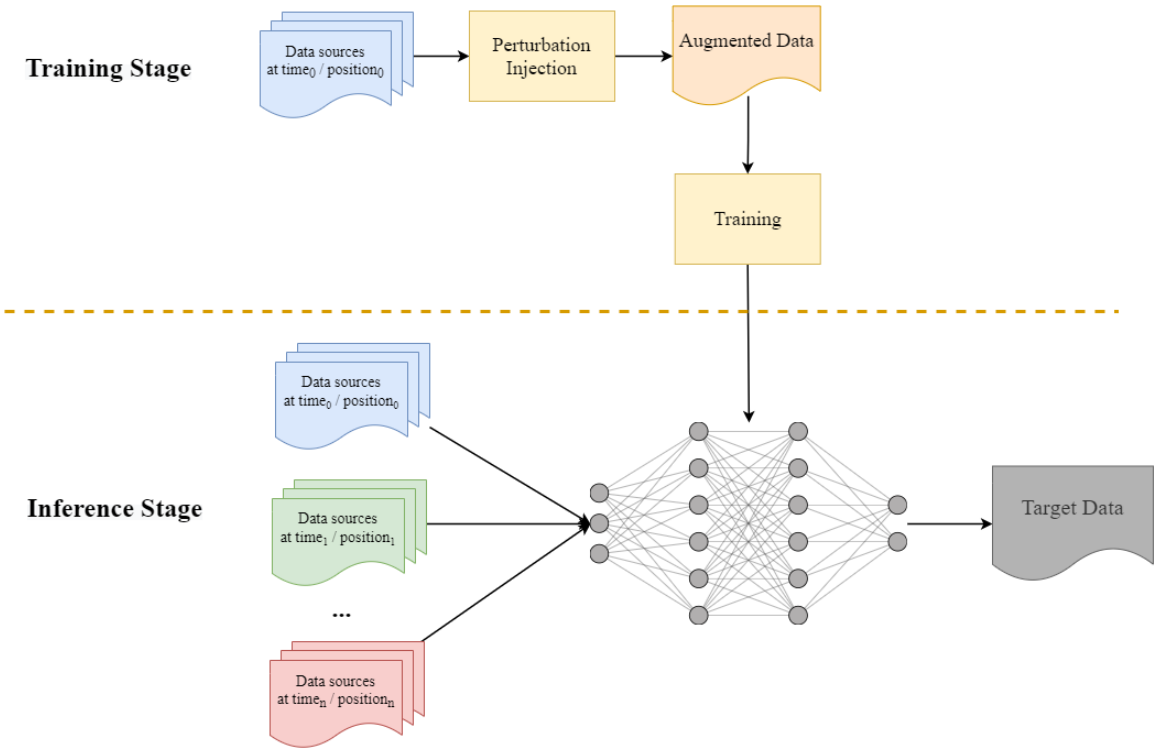


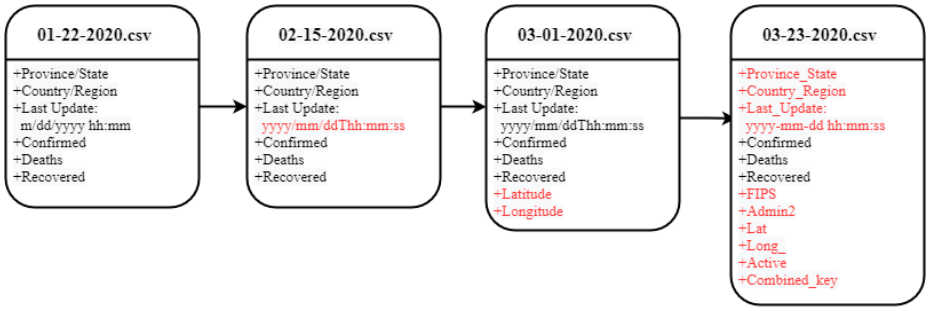
Figure 7. Overview: Data integration based on deep learning models

Note: Different timestamps or positions represent different versions of dataset that are affected by schema changes or schema variations.

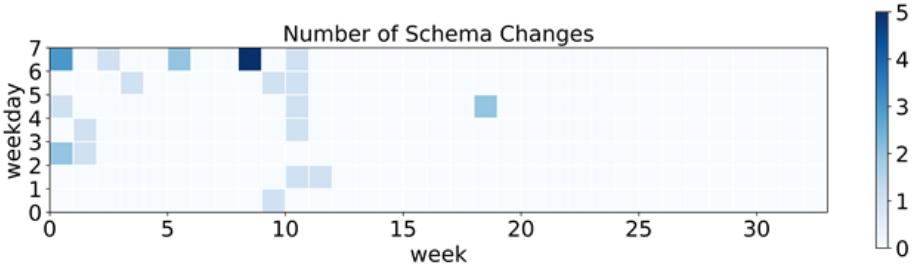
Hopkins University (JHU)[8], and the Google mobility data[9]. The former dataset records the trend of COVID-19 cases with times, sorted by the geographical locations worldwide, and the latter one mainly contains the changes of human activities during the period of COVID-19, also sorted by similar geographical information. It is intuitive to integrate these two datasets based on the information they shared but not identical. The shared attribute that exists in both datasets, geographic information, can be treated as the join key, if we describe this data integration in the context of database join operation. However, since the shared keys are not in the same level of geographical information, as well as the schema changes happen frequently in COVID-19 data repository, as shown in Figure 8, it is impractical to use a simple join operation in this scenario.

[8]https://github.com/CSSEGISandData/COVID-19

[9]https://www.google.com/covid19/mobility



(a) Examples of schema updates for COVID-19 dataset



(b) Number of schema changes that happened in COVID-19 dataset

Figure 8. During the development of COVID-19 data repository, schema changes happened frequently and continuously

Table 1 shows the basic schema of COVID-19 and Google mobility data. Due to the versioning updates and the changing of data usage, we also observed various schema evolutions include but not limited to attribute name changes (e.g., Longitude → Long_), addition and removal of attributes (e.g., from six attributes to fifteen attributes), attribute type changes (e.g., date formats), key changes (e.g., from country/region, province/state to country_region, province_state, admin2.

Table 2 shows the schema changes we covered in the COVID-19 data integration scenario. We choose the dataset with the earlier version of schema as the training samples, the dataset after several rounds of schema evolutions as the testing samples. We use this scenario to validate the effectiveness of the deep learning based approach against natural schema changes that happened in data repositories.

Table 1. Schema of COVID-19 and Google Mobility dataset

Dataset	Key (Row Identifier)	Attributes
COVID-19 Case	Province/State, Country/Region	Last Update, Confirmed, Deaths, Recovered, Latitude, Longitude
Google Mobility Data	country_region, sub_region_1, sub_region_2	retail, grocery, parks, transit, workplaces, residential

Note: The full attribute name for Google Mobility Data table:
retail: retail_retail_and_recreation_percent_change_rom_baseline;
grocery: grocery_and_pharmacy_percent_change_from_baseline;
parks: parks_percent_change_from_baseline;
transit: transit_stations_percent_change_from_baseline;
workplace: workplaces_percent_change_from_baseline;
residential: residential_percent_change_from_baseline.

Table 2. Attribute changes exist in COVID-19 data table

Training Data	Inference Data
Province/State	Province_State
Country/Region	Country_Region
/	Admin2
Last Update	Last_Update
Confirmed	Confirmed
Deaths	Deaths
Recovered	Recovered
Latitude	Lat
Longitude	Long_
Recovered	Recovered

Note: Change of attribute `Admin2` is in fact key expansion rather than attribute addition.

4.1.2 Machine Log Data Integration.

Cluster monitoring tools integrate various performance metrics of a cluster of devices by periodically running the device-local API, such as an omnipresent

“top“ command. However, when new types of platforms or devices are involved, hardware/software heterogeneity can cause problems. For example, CPU utilization is called “CPU_usage_user” in macOS, “cpu_user“ in Android, and “Cpu_us” in Linux. Therefore, the tool cannot work with these new types of systems without additional coding efforts. This problem is prevalent in machine or sensor data integration, where devices of different models and manufacturers may use different data schemas to describe similar information. We named this scenario as machine log data integration. To be specific, we collect the log data of machine performance using “top” command on three different platforms, Linux, Android and macOS. Each platform's data comes with similar content but various representations, which also impede the use of traditional rule-based data integration.

Table 3 shows the basic schema of machine log data on different platforms. The representation of key on all three datasets keeps the same, which is the timestamp of each machine log data generated. While the difference in attribute name and its corresponding value are the main challenges that need to be solved by our proposed deep learning based approach.

Table 4 compares the difference of attribute names across three platforms. In this scenario, data shared with the same timestamp need to be combined. Instead of natural schema changes that happen during the time, the schema variations exist across different machine platforms. Because of that, we chose the data generated on macOS and Linux as our training samples, and the data comes from Android to be the inference samples. The deep learning based approach also needs to prove its feasibility on the variation of schemas.

Table 3. Schema of machine log data on three platforms

Data Source	Key (Row Identifier)	Attributes
macOS	time	Load_Avg, Processes_total, Processes_running, Processes_sleeping, CPU_usage_user, CPU_usage_sys, CPU_usage_idle, PhysMem_unused, PhysMem_used
Linux	time	load_average, Tasks_total, Tasks_running, Tasks_sleeping, Tasks_stopped, Tasks_zombie, Cpu_us, Cpu_sy, Cpu_ni, Cpu_id, Cpu_wa, Cpu_hi, Cpu_si, Cpu_st, Mem_MiB_total, Mem_iB_free, Mem_MiB_used, Mem_MiB_buff, Swap_MiB_total, Swap_MiB_free, Swap_MiB_used
Android	time	Tasks_total, Tasks_running, Tasks_sleeping, Tasks_stopped, Tasks_zombie, cpu_user, cpu_sys, cpu_nice, cpu_idle, cpu_sirq, Mem_total, Mem_free, Mem_used, Mem_bufferes, Swap_total, Swap_free, Swap_used

4.2 Classification Tasks

In our proposed deep learning based approach, we model the entire data integration task as three independent classification tasks, as shown in Figure 9. Key prediction and attribute prediction are similar to traditional classification tasks, designed to classify the join keys and columns for each sample. Aggregation mode prediction is only used in COVID-19 data scenario since the keys are coming with different levels in different datasets. For example, COVID-19 dataset consists of a 2-level key including information about country and state, while the 3-level key in Google Mobility data

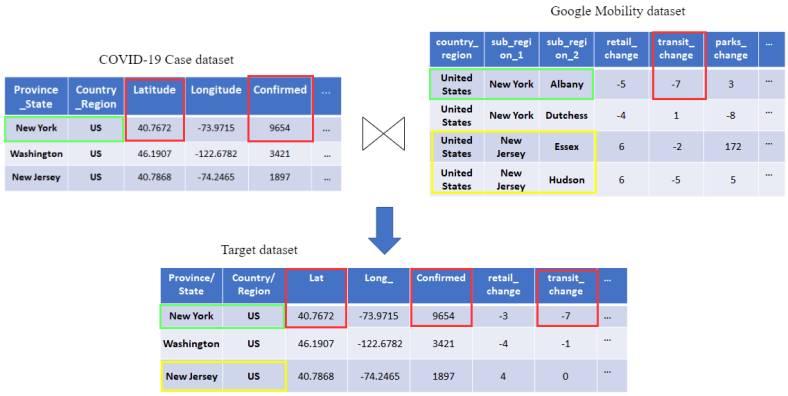
Table 4. Comparison of the different schema for machine log data

Training Data		Inference Data
macOS	Linux	Android
time	time	time
Load_Avg	load_average	/
Processes_total	Tasks_total	Tasks_total
Processes_running	Tasks_running	Tasks_running
Processes_sleeping	Tasks_sleeping	Tasks_sleeping
/	Tasks_stopped	Tasks_stopped
/	Tasks_zombie	Tasks_zombie
CPU_usage_user	Cpu_us	cpu_user
CPU_usage_sys	Cpu_sy	cpu_sys
/	Cpu_ni	cpu_nice
CPU_usage_idle	Cpu_id	cpu_idle
/	Cpu_wa	/
/	Cpu_hi	/
/	Cpu_si	cpu_sirq
/	Cpu_st	/
/	Mem_MiB_total	Mem_total
PhysMem_unused	Mem_iB_free	Mem_free
PhysMem_used	Mem_MiB_used	Mem_used
/	Mem_MiB_buff	Mem_bufferes
/	Swap_MiB_total	Swap_total
/	Swap_MiB_free	Swap_free
/	Swap_MiB_used	Swap_used

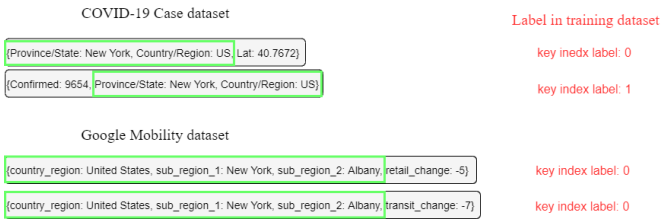
comes up with country, state and county. This requires aggregation operations when conducting the data integration task.

4.2.1 Key (Row Identifier) Prediction

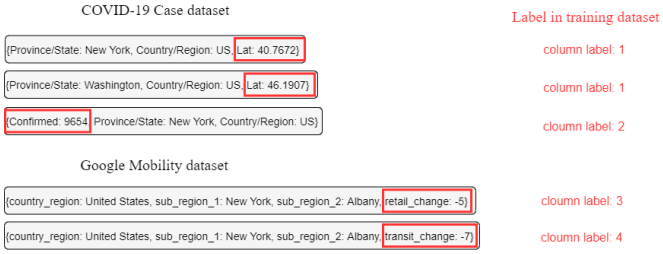
In order to complete a data integration task, it requires us to join two tables together based on their shared keys. In the context of deep learning models, we design the key prediction that helps categorize and locate the join key in source and target



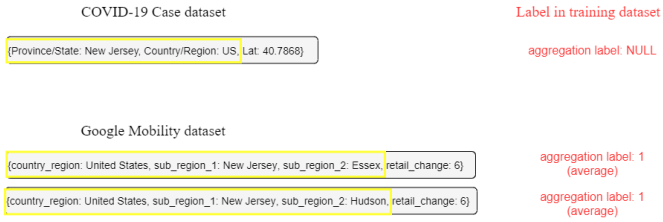
(a) Overview of data integration on COVID-19 data



(b) Example of Key Index Prediction



(c) Example of Column (attribute) Prediction



(d) Example of Aggregation Mode Prediction

Figure 9. Green frames annotate the position of keys in the table and data samples, the label represents the index of keys in the whole samples; Red frames annotate columns in the table and data samples, the label represents the different class of columns; Yellow frames annotate the rows need to be aggregated in the table and data samples, the label represents various aggregation modes (summation, average, etc.). The label shown may not be the same as the real training process.

tables. To be specific, we try to classify the key of each sample, thus the keys in each source table as well as in the target table that belong to the same class can be mapped together, even if the schema changes are involved in the source data. As we discussed before, the label representation for keys can be formatted in two ways: (1) based on the index (position) of the key in the supercell-based object. (2) based on the key value itself.

Since the values of key in machine log dataset, time, are unlimited and self-growing when new data is generated, it is impractical to use key values to identify different keys. Based on that, we will mainly take COVID-19 dataset as an example to explain the difference between these two concepts.

4.2.1.1 Key Prediction Based on Value

With this design, the label will be based solely on the content of key values, keys that represent the same content will be signed with the same label (even though their actual value might be different due to the scheme changes or variations). This value-label mapping will be learned during the training process, and further schema changes can be handled by the models trained with robustness. In this case, we utilize a cell-level abstraction of the training and testing samples, so that the input of the deep learning models is a word sequence with the format as [key: key value, attribute: attribute]. However, there exist problems in this design of prediction task. For example, the key value in the table can be continuous or self-growing, such as the datetime, growing IDs, phone number, and so on. If we take these keys as training labels, it causes poor accuracy, as the testing labels will never be seen in the training

process. To solve the problem, for the key label, we provide another option that uses the position of the key in the source cell features to indicate it.

4.2.1.2 Key Prediction Based on Index

In the situation that the number of classes for keys in samples could be too large, or self-grew when new data participated in the data integration, the key-label mapping based on value is unfeasible. In this design, we categorize the keys with two steps. At first, instead of directly classifying the key itself, we care more about the position of the key among the whole sample with the supercell-level abstraction. After we identify the right position of the key, a parsing phase is used to map the key from the source to target table. If the representation of keys keeps the same between source and target, the mapping operation happens naturally, based on key values. Otherwise, a similarity-based approach can be used to pair the key. To be specific, we embed the text sequence of keys into vectors using word/sentence embedding tools, and compute the similarity pairwise between the source keys and target keys.

In this case, we construct the input with the format as [key: key value, attribute 1: attribute 1 value, attribute 2: attribute 2 value, ...].

4.2.2 Attribute (Column) Prediction

The key prediction is designed to handle the scenario that two tables have the same key values, no matter what the attributes are, which is more like the join operation. In addition, we design the attribute prediction task to simulate the union operation

so that the same attribute in two source tables can be mapped together in the target table.

Since the number of attributes is limited in the source or target dataset, we will simply predict the attribute based on its value. The attributes that are newly added from the schema change and were not shown in the target table, will be taken as discard.

We model the attribute prediction task with two different data granularities. For supercell-level data abstraction, one sample covers more than one attribute, thus the prediction becomes a multi-label classification task, while for cell-level data abstraction, it is a simpler single-label classification since one sample (cell) only contains one attribute. We will discuss the difference between these two abstractions shown in experiment results in the next chapter.

4.2.3 Aggregation Operation Prediction

One of the schema changes we observed in COVID-19 dataset is key expansion. To be specific, the geographic information that forms the key of COVID-19 dataset comes with multiple levels and the number of levels changes during the time. At the initial schema of COVID-19 dataset, only nation/region level and state/province level information is covered, while during the version updating, a lower level of county information is added in the table as a part of the key. In order to integrate data with key expansion changes, the aggregation operation needs to be done so that all the rows with low-level counties belonging to the same high-level state will be aggregated as one. The aggregation operations vary depending on the context of attributes. For example, the operation for attribute `Confirmed` is an addition, since

all the confirmed cases under counties need to be added together to get the confirmed cases for the whole state. On the other hand, the operation for attribute `residents` is average, since it represents the percentage change of resident activity. The aggregation information for each dataset and attribute is extracted from: (1) We parse the data integration application code with join operations to get the information including whether the aggregation exists in this scenario or not, as well as which aggregation mode should be chosen; (2) We specify the aggregation information together with the source dataset, thus when an incoming integration request requires such source dataset, the aggregation prediction can be done. We further propose to augment the training data with hierarchical relationships, we will discuss the details of it in the data augmentation part.

The data abstraction for aggregation prediction task is based on cell-level, the input is formalized with the same format of key value prediction: [key: key value, attribute: attribute value].

4.2.4 Performance Acceleration

The above-mentioned approach requires multiple inferences over each cell, which incurs significant overhead. However, we observe that many open datasets are in certain formats so that each row or record has the same structure, and the key is always in the same position, such as CSV files, or performance logs. Therefore, we only need to parse and make column identifier inferences on the cells of one tuple, while the learned mapping is directly applicable to the cells in other tuples. Furthermore, the key index prediction only needs to be performed on one cell, as all cells in the same tuple share the same key. This will significantly reduce the overhead.

4.3 Data Augmentation

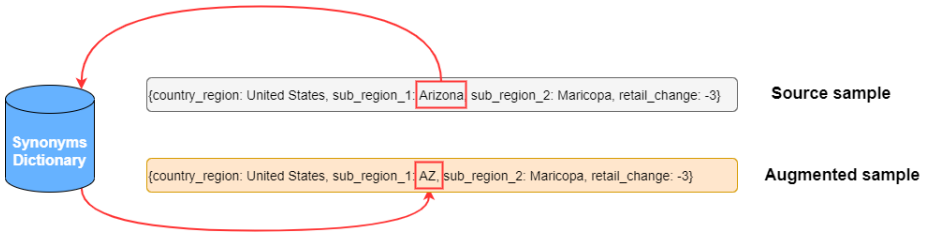
Automatically searching random perturbations to represent potential schema changes and adding these perturbations to augment the training data is an effective way of increasing the amount of training data and boosting model performance. Inspired by the progress that data augmentation achieved in NLP area (Feng et al. 2021) (Wei and Zou 2019), we propose to add specially designed perturbations to our training data, to improve the robustness of training models to schema changes.

Each of our training samples consists of several attribute name-attribute value pairs that describe one unit of the data abstraction (object, supercell and cell), and form as a plain sentence. To address the **attribute name change** and **value format change** challenges, We propose: (1) for each sentence randomly sampled following a uniform probability distribution, to add new sentences by replacing the word using synonyms extracted from synonyms dictionaries, such as Thesaurus (Zhang, Zhao, and LeCun 2015) and customized synonym dictionary based on expert's domain knowledge; (2) for each word randomly sampled, to diversify the form of words leveraging the stemming and prefixing words generated by Natural Language Toolkit (NLTK)[10]. To address the **key ordering change**, for each training sample randomly sampled, we randomly change the position of the key in the feature vector. To address the **key expansion change**, in the features for each training sample that is randomly sampled, we look for a categorical attribute that can be expanded into an array of sub-level concepts (e.g., Arizona is flattened into an array of counties in Arizona) leveraging the hierarchical relationships in ontology databases such as Google Knowledge Graph[11],

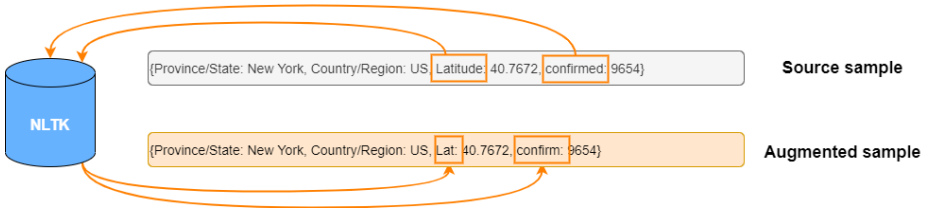
[10]https://www.nltk.org

[11]https://developers.google.com/knowledge-graph

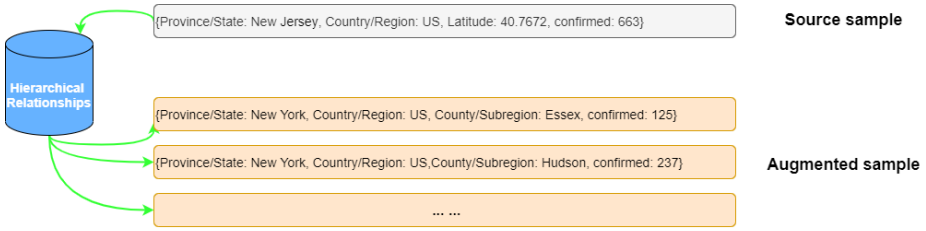
then we add new training samples by flattening this training sample into a list of sub-level training samples with summation aggregation function (e.g., the numbers of confirmed, death, and recovered COVID-19 cases of all counties belonging to Arizona need to be summed up respectively before being mapped to the target table). For example, a source training sample ("AZ", "confirmed", 33) will be replaced by a set of new training examples such as ("AZ", "Maricopa", "confirmed", 13, add) and ("AZ", "Pima", "confirmed", 20, add) in the training data as perturbations. Figure 10 shows several data augmentation examples.



(a) Synonyms Replace



(b) Word Stemming and Prefix



(c) Hierarchical Relationship

Figure 10. Examples of data augmentations added to source dataset

Chapter 5

EVALUATION AND EXPERIMENT RESULTS

5.1 Environment Setup

Our evaluation focuses on two unmanaged data integration scenarios: COVID-19 data integration, and machine log integration, as we illustrated in Chapter 5. We train and test the Transformer model with PyTorch[12], and the Bi-LSTM model with TensorFlow [13], both using a Google Colab[14] instance that has installed one P100 GPU with 25.5 GB GPU memory. The key value/index predictions are trained with 50 epochs, while the column/aggregation action predictions are trained with 60 epochs.

5.2 Model Training and Hyperparameters

5.2.1 Training and Testing Dataset Construction

In order to construct the training and testing dataset we need in the experiment, we first extract the source data from JHU COVID-19 repository and Google Mobility data website, both of them are CSV files. In addition, we generate the machine log data from three different operating systems, Linux, macOS and Android, which are

[12]https://pytorch.org/

[13]https://www.tensorflow.org/

[14]https://colab.research.google.com/notebooks/intro.ipynb

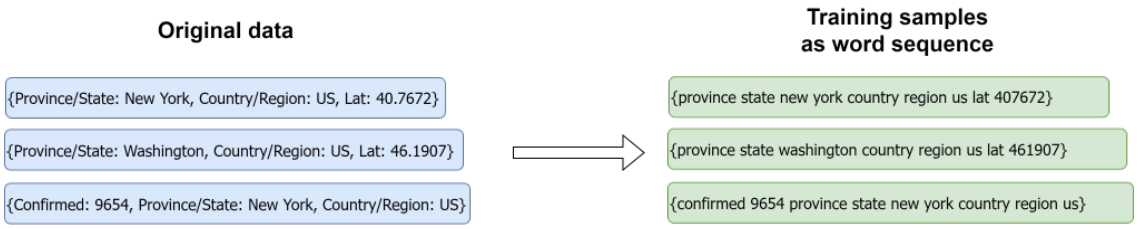


Figure 11. We further parse the original data through the segmentation and data cleaning stage to construct the training dataset.

Note: The numerical values are kept in COVID-19 dataset since they were only a small proportion. While we remove all the numerical values in Machine log dataset as their existences affect the results.

all plain text. We manually parse these data to tabular format with clearly specified schema to construct the source tables and target tables in each scenario.

Since the original data extracted from the source data tables contain irrelevant symbols and contents like dash, slash, numerical numbers and so on, we parse the source and target data through a data segmentation and cleaning phase so that both the training and testing data are formed by the word sequences. Figure 11 shows an example that parses the original samples to training samples based on word sequences.

After we parse the data samples to the format of word sequence, we utilize data augmentation to inject perturbations into the training dataset. We already explained the construction of our data augmentation approaches in the previous content, thus we will not cover the details here.

For Bi-LSTM model, we utilize a fastText model that pretrained on a larger source data (for example, we use the COVID-19 data generated from March 1st, 2020 to March 20th, 2020 as the training corpus for fastText model), as the embedding layer that converts the word sequence to numerical vectors. For Transformer model, a

pretrained word tokenizer [15] is used to convert the word sequence to tokens, and the token sequences are further handled by the model to learn a reasonable vector representation. The noteworthy thing here is, since both the word embedding and tokenization tools include the mechanism of Out-Of-Vocabulary (OOV) words, not all words in the original samples are converted to the vectors or tokens, some numerical values, as well as the uncommon words that only exist a few times, will be taken as noises during the converting and learning process.

In summary, the training data for COVID-19 integration scenario, includes 33, 174 samples collected from the JHU COVID-19 data on March 20th, 2020, while the testing data includes 49, 410 collected on later dates of March 23rd, 2020. The training data for machine log integration scenario, includes 60, 000 samples collected from the Linux and macOS platform, while the testing data includes 17, 000 collected from the Android platform.

5.2.1.1 Data Labeling

For each independent prediction task, we construct an independent label space based on the dataset. Table 5 illustrates the number of labels assigned to COVID-19 and Machine Log data integration scenarios. Key-index prediction is built based on the supercell-level sample abstraction, the length of supercell used in COVID-19 dataset is 6, and 8 in Machine Log data. COVID-19 dataset contains 2804 different key values, thus the number of labels based on key values is 2804. Besides, COVID-19 dataset and Machine Log dataset covers 12 and 21 attributes respectively. Only 2 aggregation modes, summation and average, are included in the COVID-19 dataset.

[15] https://huggingface.co/bert-base-uncased

Table 5. Number of labels assigned to COVID-19 and Machine log training dataset, with different prediction tasks

Prediction Task	COVID-19 dataset	Machine Log dataset
index-based key prediction	6	8
value-based key prediction	2804	/
column prediction	12	21
aggregation mode prediction	2	/

Note: The value-based key and aggregation mode prediction are not applicable to Machine Log dataset.

5.2.2 Model Description

For the experiment tested on Bi-LSTM model, we choose to embed the input samples using a pretrained fastText model under skip-gram mode, with the embedding size of 150. For both forward and backward LSTM layers, the hidden size is chosen as 512, and the hidden size of 256 for the following fully connected layer. The model was trained using Adam Optimizer (Kingma and Ba 2014) and Dropout (Srivastava et al. 2014) with the rate $\rho = 0.8$. The model training is tuned with the learning rate $\eta = 0.0001$ and batch size $b = 64$.

For Transformer model, we adopted the pretrained tokenizer from BERT ('bert-base-uncased'). We construct our transformer model using 12 encoder blocks, for each with 8 attention heads, and the hidden size of 128 (L=12, H=128, A=8). The model was also trained using Adam Optimizer and Dropout with rate $\rho = 0.9$. We set the learning rate with $\eta = 0.0001$ as well as the batch size with $b = 64$

For each data integration task, we used three independent but identical Bi-LSTM and Transformer models to train three independent prediction tasks (for Machine Log dataset we just use two).

Table 6. Testing accuracy using Transformer model, without data augmentations

Task	Accuracy	F1 Score	Precision	Recall
COVID-19: index-based key prediction	87.18%	0.8743	0.9057	0.9394
COVID-19: value-based key prediction	100%	1	1	1
COVID-19: column prediction	100%	1	1	1
COVID-19: aggregation operation prediction	100%	1	1	1
Machine-log: index-based key prediction	93.32%	93.32%	93.32%	93.32%
Machine-log: value-based key prediction	/	/	/	/
Machine-log: column prediction	82.35%	0.8235	0.8235	0.8235

Note: The value-based key prediction is not applicable due to the self-growing key. The accuracy is computed based on cell-level samples, rather than object-level. All the metrics are calulated using the 'macro' average method

Table **??** shows the experiment results get on Transformer and Table 7 shows the results on Bi-LSTM, both are generated without any data augmentation injected in the training dataset.

We compute all the metrics including accuracy only based on the cell-level samples, which means that

$$\text{accuracy} = \frac{\text{correct predicted cells}}{\text{total number of cells in the table}}$$

On the other hand, accuracy can also be computed based on object-level (row-level) samples, which means that only if all cells in one object are predicted correctly, the object will be counted as correct.

By comparing the results get from two models, we discover that the transformer model that makes use of attention mechanism can achieve better accuracy as well as other performance metrics than the Bi-LSTM model in column prediction tasks for the COVID-19 case, and better results in key predictions for machine log case. On the other hand, the results demonstrate the effectiveness of training separate models to predict row index labels, column identifier labels, and aggregation actions respectively. They also verify our assumption that it is possible to use deep learning models to solve the data integration process.

Table 7. Testing accuracy using Bi-LSTM model, without data augmentations

Task	Accuracy	F1 Score	Precision	Recall
COVID-19: index-based key prediction	87.47%	0.8747	0.8747	0.8747
COVID-19: value-based key prediction	88.46%	0.8846	0.8846	0.8846
COVID-19: column prediction	86.10%	0.861	0.861	0.861
COVID-19: aggregation operation prediction	100%	1	1	1
Machine-log: index-based key prediction	45.15%	0.4515	0.4515	0.4515
Machine-log: value-based key prediction	/	/	/	/
Machine-log: column prediction	82.35%	0.8235	0.8235	0.8235

Note: the value-based key prediction is not applicable due to the self-growing key.

5.3.1 Comparison of Results on Different Granularity of Data Abstraction

We proposed to abstract the data samples in different granularities in the previous chapter. In the experiment part, We evaluate the results using two different levels of data granularity, cell abstraction and supercell abstraction.

Table 8. Testing accuracy on Transformer model, based on different data granularity

Task	Accuracy	F1 Score	Precision	Recall
COVID-19:column prediction (supercell based)	95.55%	0.9864	0.9887	0.9841
COVID-19:column prediction (cell based)	100%	1	1	1
Machine-log:column prediction (supercell based)	50.1%	0.7344	0.792	0.6844
Machine-log:column prediction (cell based)	82.35%	0.8235	0.8235	0.8235

Table 8 and Table 9 compare the two abstractions on performance and time overhead, trained on Transformer model.

Table 9. Time overhead on Transformer model, based on different data granularity

Task	Training Time	Inference Time
COVID-19:column prediction (supercell based)	6 min 21 s	3.4 s
COVID-19:column prediction (cell based)	40 min 12 s	24 s
Machine-log:column prediction (supercell based)	18 min 43 s	2 s
Machine-log:column prediction (cell based)	37 min 40 s	10 s

Note: Inference time is measured by inferencing the whole table.

Table 10. Testing accuracy on Bi-LSTM model, based on different data granularity

Task	Accuracy	F1 Score	Precision	Recall
COVID-19:column prediction (supercell based)	81.50%	0.8745	0.8774	0.8716
COVID-19:column prediction (cell based)	86.10%	0.8610	0.8610	0.8610
Machine-log:column prediction (supercell based)	40.05%	0.6875	0.7353	0.6455
Machine-log:column prediction (cell based)	82.35%	0.8235	0.8235	0.8235

Table 10 and Table 11 show the comparison results that trained on Bi-LSTM model.

With supercell abstraction, one sample covers multiple attributes and their value, thus the column prediction becomes a multi-label prediction task, and that causes model performance loss, compare to single label prediction within cell abstraction.

On the other hand, since one sample cover more attributes, the number of samples parsed as the training dataset is less, the time overhead for model training and inference is less for supercell abstraction. In certain circumstances, we can make a tradeoff between performance and time overhead. For example, when the inference latency is a priority in the data integration process, we choose supercell abstraction to parse the training and inference data.

5.3.2 Performance Improvement with Data Augmentation

In COVID-19 dataset scenario, we use testing data extracted from the real-world JHU database with four types of schema changes: attribute name changes, attribute value format changes, key expansion, and key ordering changes. For the machine

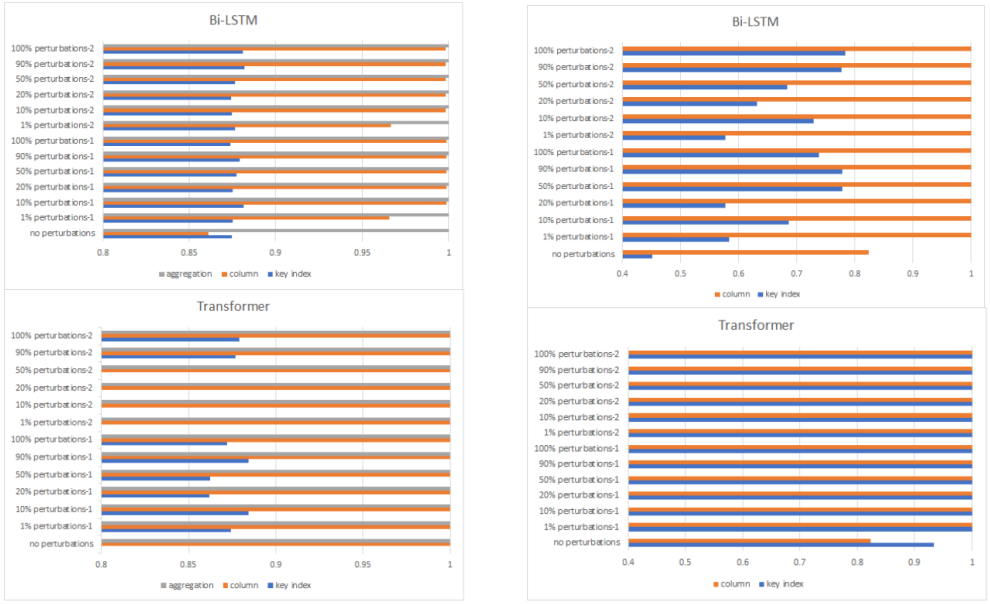
Table 11. Time overhead on Bi-LSTM model, based on different data granularity

Task	Training Time	Inference Time
COVID-19:column prediction (supercell based)	3 min 42 s	3.7 s
COVID-19:column prediction (cell based)	18 min 29 s	18.4 s
Machine-log:column prediction (supercell based)	2 min 27 s	4.1 s
Machine-log:column prediction (cell based)	5 min 37 s	6.3 s

log scenario, the testing data only contains attribute name changes, attribute value format changes, and key ordering changes. For both evaluation scenarios, we observed significant accuracy improvement for handling schema changes by augmenting the training data (w/ perturbation-1 and w/perturbation-2) as illustrated in Figure 12, compared to the model trained without perturbations. Perturbation-1 is based on synonyms extracted from Google knowledge graph, NLTK stemming, and prefix; Perturbation-2 is based on synonyms extracted from a customized domain-specific dictionary, NLTK stemming, and prefix. The percentage of perturbations represents the sampling probability in the training dataset.

5.4 Extra Study

We have also done some extra studies to compare and thus improve the performance of deep learning based data integration. We investigated to model data integration as other types of prediction tasks, as well as using other approaches to improve the model robustness.



(a) COVID-19 Data (b) Machine Log data

Figure 12. We added the perturbation with 6 different percentage, 1%, 10%, 20%, 50%, 90% and 100%. The results are also compared to the group without perturbations added.

5.4.1 Next Word Prediction Based on Pretrained Bert Model

Next word prediction is a task that proposed in the published paper of BERT (Devlin et al. 2018), the task is designed to take more consideration about the relationship and context between words, when training the language model, thus improving the model performance.

In the training process, several words will be masked out as `[mask]` randomly in the training dataset, and the model is required to predict every `[mask]` based on its context, position and so on. In general, it tries to recover an incomplete text by predicting all the masked out words.

In the data integration task, we modeled one of the column prediction tasks as the next word prediction task, where during the training process, we add the

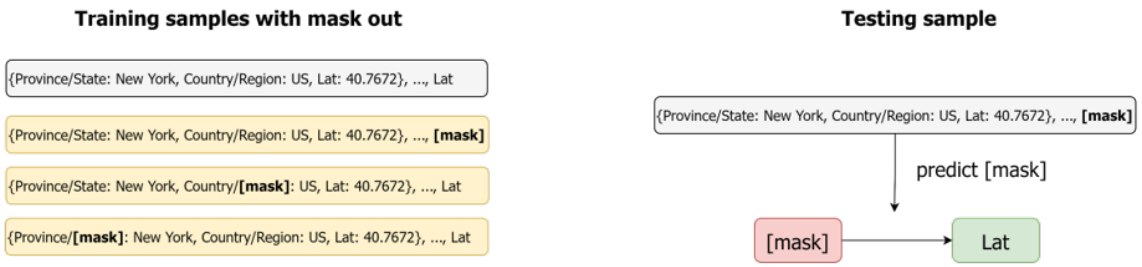


Figure 13. Modeling column prediction as next word prediction task. The training dataset is augmented with masked out samples. In inference stage, testing samples with [mask] is being predicted to get the real attribute name.

attribute name at a fixed position of the whole sentence (i.e, as the last word in the sentence), and we augment the training dataset by randomly replacing words including the attribute name to `[mask]`. After the model training process, the `[mask]` corresponding to attribute names will be predicted correctly. Figure 13 illustrates the modeling of next word prediction task.

The result is shown in Table 12. However, this method can not be generalized. In our training samples, the number of different words, or the text richness, is limited, which means the label space for predicting the `[mask]` is limited, this makes the task of next word prediction on our dataset easier and less powerful, compared to the natural English text. Due to the time limit, we will leave this investigation as future work.

Table 12. Testing accuracy for Next Word Prediction on Transformer Model

Task	Accuracy
Machine-log:column prediction (cell based)	82.53%

5.4.2 Boosting Model Robustness using Adversarial Learning

Adversarial learning is first proposed in Computer Vision area to boost the model robustness against malicious attacks or erroneous samples (Goodfellow, Shlens, and Szegedy 2014). The original training samples were added with adversarial perturbations, before they got input into the training process, and thus making the models more robust to malicious attacks or noisy samples.

Recently, massive works investigated the effectiveness of adversarial learning on NLP tasks (Miyato, Dai, and Goodfellow 2016) (Sato et al. 2018) (Michel et al. 2019). Since in computer vision, the image is represented by the value of each pixel and the adversarial perturbations can be simply a few pixels' values that are added to the whole image. While this is very different in the context of NLP, where training samples are usually plain text that consists of words, numbers and symbols. In order to add adversarial perturbations on NLP dataset, there are two popular methods:

- Text data can be converted into vectors through an embedding layer before they got into the model. The adversarial perturbations can be generated based on the embedded text vectors, just like it does in image pixel values.
- The perturbation will only be considered at the higher level of words, in which the words in the text can be replaced by others, so that the prediction results change completely. This requires an efficient mechanism to search the right perturbations from the word space, that maximize the loss function. Compare to the embedding perturbations, this method keeps the semantic context since the perturbations added will always correspond to real words.

On the other hand, since text samples are embedded into value vectors before they got into the models, the perturbations can also be generated based on the embedding

vectors, the easiest way is adding a regularization on each embedding vector, so that the value keeps in a reasonable range. However, this method does not consider the semantic context during each perturbation, since the perturbed samples may no longer correspond to a real word or number. For the data integration task, we utilized the former method to generate adversarial perturbations based on the embedding vector of training samples.

Adversarial learning can be seen as a Min-Max optimization problem. First, we try to identify the worst as well as the optimal perturbation $\boldsymbol{r} + adv$ that maximizes the loss function $\leq$, as shown in the following equation:

$$\boldsymbol{r}_{adv} = \underset{\boldsymbol{r} \leq \epsilon}{\operatorname{argmax}}[\ell(X_{+\boldsymbol{r}}, Y, \mathcal{W})] \tag{5.1}$$

where $\boldsymbol{r}$ is the perturbation added to input $\boldsymbol{X}$, and $\mathcal{W}$ is the fixed weight of the model. ϵ is a tunable parameter that limits the norm of perturbation $\boldsymbol{r}$. However, it is infeasible to calculate or even estimate $\boldsymbol{r}_{adv}$ using Equation 5.1 directly. Instead, we can only approximate the value by using the following equations:

$$\boldsymbol{r}_{adv} = \epsilon \frac{\boldsymbol{g}}{||\boldsymbol{g}||_2}, \quad \boldsymbol{g} = \nabla_{\boldsymbol{w}} \ell(X, Y, \mathcal{W}) \tag{5.2}$$

where $\boldsymbol{g}$ is the gradient of the loss function ℓ.

The perturbation sample is generated by $X_{adv} = X + bmr_{adv}$, which is similar to the scenario of schema changes that happened in the original data. On the other hand, the perturbation bmr_{adv} is bounded by parameter ϵ, and the optimal value of ϵ can be tuned by approximating the difference between original samples S_o and schema changed samples S_p. We will use V_o and V_p to represent the embedding vector for each of them.

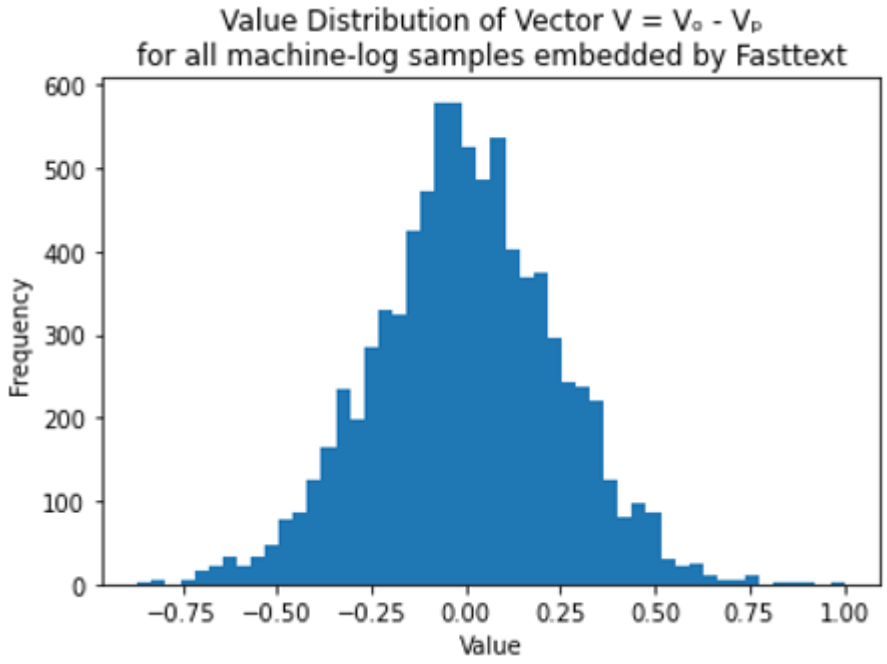


Figure 14. Value distribution of the difference between original samples and schema changed samples

We calculated the difference value $V = V_o - V_p$ between original samples and schema changed samples, the value distribution of the vector V is shown in Figure 14. Since 95% of the value are distributed in the interval of [-0.476, 0.476], we set $\epsilon = 0.476$ in our experiment of adversarial learning.

We adopted the Bi-LSTM model with the same parameters, we trained the model with column prediction on Machine Log data. The result is shown in Table 13. The result is not satisfied, since we didn't do much tuning with other hyperparameters, as well as the problem of adversarial learning itself. We will also leave this as future work.

Table 13. Testing accuracy on Bi-LSTM Model with Adversarial Perturbations

Task	Accuracy	F1 Score	Precision	Recall
Machine-log:column prediction (cell-based)	0.1176	0.1176	0.1176	0.1176

Chapter 6

CONCLUSION AND FUTURE WORKS

6.1 Summary

In this paper, we investigated the data integration problem with schema changes based on deep learning models. In particular, we studied the problem with the assumption that, the source datasets are or can be converted to tabular-format data, and the target table will be specified with clear schema by each integration request. We utilized a cell-based representation to abstract the tabular data as data samples with labels that can be effectively learned by the deep learning models. We conducted experiments on two different data abstractions: cell-level abstraction and supercell-level abstraction. We also summarized the common types of schema changes existed in the real-world data sources that interrupt the data integration task. We proposed a novel way to model the data integration task as several prediction tasks, including key prediction, attribute prediction and aggregation mode prediction. Besides that, we proposed to use two deep learning models, Bi-LSTM and Transformer to validate the effectiveness of our approach. We further made the trained model more robust to potential schema changes by leveraging data augmentation and adversarial learning, thus the data integration pipeline won't be interrupted frequently in the near future. Our proposed approach works well on two real-world data integration scenarios, each of which covers light-weighted schema changes samples.

6.1.1 Limitations

Our current work still has many limitations, including but not limited to: (1) The adaptivity to various raw data formats. Our current approach is built on the hypothesis that both source data and target data should be tabular format or can be converted to tabular format with clearly defined schemas. This requires extra human labors to check the source data and execute the conversion step; (2) Human efforts are needed to generate the training dataset. The training dataset is constructed manually for every data integration task, it is not generalized enough at this moment; (3)Limitations in our proposed key prediction tasks. The value-based key prediction cannot work with the self-growing key, and has a poor performance when the label space of key values is huge. The indexed-based key prediction approach cannot work with the schema change of new column addition, since after the addition, the index of key goes beyond the trained index spaces; (4) Model robustness with schema changes. Our current testing scenarios only cover limited schema changes, which may not be adaptable to all applications; (5) Performance comparison to attribute-based data abstraction. We observed the existence of attribute-based data abstraction in our source dataset, which might be more effective in predicting column mappings than the cell-based data abstraction, while we did not investigate deeply with this approach.

The work included in this thesis is a part of a project that aims to develop a fully automatic end-to-end data integration system. We didn't cover works like automatic training sample generation, model reusing, and so on in this thesis. We believe that the conclusion got from this work can be a guideline that leads to more research investigations on using deep learning models to solve the data integration problems.

In summary, our contribution in this work can be generalized as following two points, we proposed:

- A novel solution of data integration tasks based on deep learning models that significantly reduces system downtime;
- A robust modeling of current data integration processes that involve tabular data with schema changes.

6.2 Future Work

The researches on data integration have been done for years, while there is still no perfect solution that automates the whole data integration pipeline, from the initial stage of related data discovery (Miller 2018), to the final data assembly stage. We present some future works that could improve our works that have been done in this paper, as well as some promising research directions:

- In database area, a complete system is important and essential, to being recognized as a successful product. Our current work can be improved by but not limited to: (1) proposing a mechanism that manages the model training, fine-tuning and reusing; (2) adding a data parsing function after the output of model to assemble data into target dataset based on the predicted label. On the other hand, the investigation on adversarial learning can be continued. Schema changes and variations that happen in source data are unpredictable, we still need a more robust model to guarantee the accuracy of output target data.
- Supervised learning has become the past in deep learning area, recent work (Wu et al. 2021) investigates the problem of entity matching using weak supervised learning, as for supervised learning, the lack of data samples with high-quality

labels is the main bottleneck. This problem also exists in data integration tasks, which we proposed as the problem of self-growing key. It is promising to investigate unsupervised learning for data integration tasks, as the well-labeled dataset is precious and rare, while low-quality datasets will hugely impact the performance of models.

REFERENCES

Abadi, Daniel, Anastasia Ailamaki, David Andersen, Peter Bailis, Magdalena Balazinska, Philip Bernstein, Peter Boncz, Surajit Chaudhuri, Alvin Cheung, AnHai Doan, et al. 2020. "The seattle report on database research." *ACM SIGMOD Record* 48 (4): 44–53.

Alexe, Bogdan, Balder Ten Cate, Phokion G Kolaitis, and Wang-Chiew Tan. 2011. "Designing and refining schema mappings via data examples." In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data,* 133–144.

An, Yuan, Alex Borgida, Renée J Miller, and John Mylopoulos. 2007. "A semantic approach to discovering schema mapping expressions." In *2007 IEEE 23rd International Conference on Data Engineering,* 206–215. IEEE.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. 2014. "Neural machine translation by jointly learning to align and translate." *arXiv preprint arXiv:1409.0473.*

Banerjee, Jay, Won Kim, Hyoung-Joo Kim, and Henry F Korth. 1987. "Semantics and implementation of schema evolution in object-oriented databases." *ACM SIGMOD Record* 16 (3): 311–322.

Bernstein, Philip A. 2003. "Applying Model Management to Classical Meta Data Problems." In *CIDR,* 2003:209–220. Citeseer.

Bernstein, Philip A, and Erhard Rahm. 2000. "Data warehouse scenarios for model management." In *International Conference on Conceptual Modeling,* 1–15. Springer.

Bernstein, Philip A., and Sergey Melnik. 2007. "Model Management 2.0: Manipulating Richer Mappings." In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data,* 1–12. SIGMOD '07. Beijing, China: Association for Computing Machinery. https://doi.org/10.1145/1247480.1247482.

Bojanowski, Piotr, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. "Enriching Word Vectors with Subword Information." *arXiv preprint arXiv:1607.04606.*

Brown, Tom B, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. "Language models are few-shot learners." *arXiv preprint arXiv:2005.14165.*

Campbell, Hamish A, Ferdi Urbano, Sarah Davidson, Holger Dettki, and Francesca Cagnacci. 2016. "A plea for standards in reporting data collected by animal-borne electronic devices." *Animal Biotelemetry* 4 (1): 1–4.

Cho, Kyunghyun, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." *arXiv preprint arXiv:1406.1078.*

Christen, P. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection. 2012.*

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

Curino, Carlo, Hyun Jin Moon, Alin Deutsch, and Carlo Zaniolo. 2013. "Automating the database schema evolution process." *The VLDB Journal* 22 (1): 73–98.

Curino, Carlo A, Hyun J Moon, and Carlo Zaniolo. 2008. "Graceful database schema evolution: the prism workbench." *Proceedings of the VLDB Endowment* 1 (1): 761–772.

Curino, Carlo A, Letizia Tanca, Hyun J Moon, and Carlo Zaniolo. 2008. "Schema evolution in wikipedia: toward a web information system benchmark." In *In International Conference on Enterprise Information Systems (ICEIS.* Citeseer.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805.*

Ding, Jialin, Umar Farooq Minhas, Jia Yu, Chi Wang, Jaeyoung Do, Yinan Li, Hantian Zhang, Badrish Chandramouli, Johannes Gehrke, Donald Kossmann, et al. 2020. "ALEX: an updatable adaptive learned index." In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data,* 969–984.

DiScala, Michael, and Daniel J Abadi. 2016. "Automatic generation of normalized relational schemas from nested key-value data." In *Proceedings of the 2016 International Conference on Management of Data,* 295–310.

Doan, AnHai, and Alon Y Halevy. 2005. "Semantic integration research in the database community: A brief survey." *AI magazine* 26 (1): 83–83.

Ebraheem, Muhammad, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. 2017. "DeepER–Deep Entity Resolution." *arXiv preprint arXiv:1710.00597.*

Edunov, Sergey, Myle Ott, Michael Auli, and David Grangier. 2018. "Understanding back-translation at scale." *arXiv preprint arXiv:1808.09381.*

Elman, Jeffrey L. 1990. "Finding structure in time." *Cognitive science* 14 (2): 179–211.

Fagin, Ronald, Laura M Haas, Mauricio Hernández, Renée J Miller, Lucian Popa, and Yannis Velegrakis. 2009. "Clio: Schema mapping creation and data exchange." In *Conceptual modeling: foundations and applications,* 198–236. Springer.

Feng, Steven Y, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, Teruko Mitamura, and Eduard Hovy. 2021. "A survey of data augmentation approaches for nlp." *arXiv preprint arXiv:2105.03075.*

Fernandez, Raul Castro, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. "Aurum: A data discovery system." In *2018 IEEE 34th International Conference on Data Engineering (ICDE),* 1001–1012. IEEE.

Fernandez, Raul Castro, Essam Mansour, Abdulhakim A Qahtan, Ahmed Elmagarmid, Ihab Ilyas, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2018. "Seeping semantics: Linking datasets using word embeddings for data discovery." In *2018 IEEE 34th International Conference on Data Engineering (ICDE),* 989–1000. IEEE.

Goldberg, Yoav. 2016. "A primer on neural network models for natural language processing." *Journal of Artificial Intelligence Research* 57:345–420.

Goodfellow, Ian J, Jonathon Shlens, and Christian Szegedy. 2014. "Explaining and harnessing adversarial examples." *arXiv preprint arXiv:1412.6572.*

Gottlob, Georg, and Pierre Senellart. 2010. "Schema mapping discovery from data instances." *Journal of the ACM (JACM)* 57 (2): 1–37.

Heimbigner, Dennis, and Dennis McLeod. 1985. "A federated architecture for information management." *ACM Transactions on Information Systems (TOIS)* 3 (3): 253–278.

Hillenbrand, Andrea, Maksym Levchenko, Uta Störl, Stefanie Scherzinger, and Meike Klettke. 2019. "MigCast: putting a price tag on data model evolution in NoSQL

data stores." In *Proceedings of the 2019 International Conference on Management of Data,* 1925–1928.

Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. "Long short-term memory." *Neural computation* 9 (8): 1735–1780.

Holubová, Irena, Meike Klettke, and Uta Störl. 2019. "Evolution management of multi-model data." In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare,* 139–153. Springer.

Howard, Jeremy, and Sebastian Ruder. 2018. "Universal language model fine-tuning for text classification." *arXiv preprint arXiv:1801.06146.*

Howard, Philip. 2011. "Data Migration, 2011." *Bloor Research, London, UK.*

Kasai, Jungo, Kun Qian, Sairam Gurajada, Yunyao Li, and Lucian Popa. 2019. "Low-resource deep entity resolution with transfer and active learning." *arXiv preprint arXiv:1906.08042.*

Kimmig, Angelika, Alex Memory, Renee J Miller, and Lise Getoor. 2018. "A collective, probabilistic approach to schema mapping using diverse noisy evidence." *IEEE Transactions on Knowledge and Data Engineering* 31 (8): 1426–1439.

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

Konda, Pradap Venkatramanan. 2018. *Magellan: Toward building entity matching management systems.* The University of Wisconsin-Madison.

Kraska, Tim, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. "The case for learned index structures." In *Proceedings of the 2018 International Conference on Management of Data,* 489–504.

Krishnan, Sanjay, Zongheng Yang, Ken Goldberg, Joseph Hellerstein, and Ion Stoica. 2018. "Learning to optimize join queries with deep reinforcement learning." *arXiv preprint arXiv:1808.03196.*

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. 2015. "Deep learning." *nature* 521 (7553): 436–444.

Li, Guoliang, Xuanhe Zhou, Shifu Li, and Bo Gao. 2019. "Qtune: A query-aware database tuning system with deep reinforcement learning." *Proceedings of the VLDB Endowment* 12 (12): 2118–2130.

Liu, Xiaodong, Kevin Duh, Liyuan Liu, and Jianfeng Gao. 2020. "Very deep transformers for neural machine translation." *arXiv preprint arXiv:2008.07772.*

Liu, Yinhan, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. "Roberta: A robustly optimized bert pretraining approach." *arXiv preprint arXiv:1907.11692.*

Luong, Minh-Thang, Hieu Pham, and Christopher D Manning. 2015. "Effective approaches to attention-based neural machine translation." *arXiv preprint arXiv:1508.04025.*

Mao, Xian-Ling, Bo-Si Feng, Yi-Jing Hao, Liqiang Nie, Heyan Huang, and Guihua Wen. 2017. "S2JSD-LSH: A locality-sensitive hashing schema for probability distributions." In *Proceedings of the AAAI Conference on Artificial Intelligence,* vol. 31. 1.

Michel, Paul, Xian Li, Graham Neubig, and Juan Miguel Pino. 2019. "On evaluation of adversarial perturbations for sequence-to-sequence models." *arXiv preprint arXiv:1903.06620.*

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. "Efficient estimation of word representations in vector space." *arXiv preprint arXiv:1301.3781.*

Miller, Renée J. 2018. "Open data integration." *Proceedings of the VLDB Endowment* 11 (12): 2130–2139.

Miller, Renée J., Laura M. Haas, and Mauricio A. Hernández. 2000. "Schema Mapping as Query Discovery." In *Proceedings of the 26th International Conference on Very Large Data Bases,* 77–88. VLDB '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Mior, Michael Joseph, Kenneth Salem, Ashraf Aboulnaga, and Rui Liu. 2017. "NoSE: Schema design for NoSQL applications." *IEEE Transactions on Knowledge and Data Engineering* 29 (10): 2275–2289.

Miyato, Takeru, Andrew M Dai, and Ian Goodfellow. 2016. "Adversarial training methods for semi-supervised text classification." *arXiv preprint arXiv:1605.07725.*

Moon, Hyun J, Carlo A Curino, Alin Deutsch, Chien-Yi Hou, and Carlo Zaniolo. 2008. "Managing and querying transaction-time databases under schema evolution." *Proceedings of the VLDB Endowment* 1 (1): 882–895.

Moon, Hyun J, Carlo A Curino, Myungwon Ham, and Carlo Zaniolo. 2009. "PRIMA: archiving and querying historical data with evolving schemas." In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data,* 1019–1022.

Moro, Mirella M, Susan Malaika, and Lipyeow Lim. 2007. "Preserving XML queries during schema evolution." In *Proceedings of the 16th international conference on World Wide Web,* 1341–1342.

Mudgal, Sidharth, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. "Deep learning for entity matching: A design space exploration." In *Proceedings of the 2018 International Conference on Management of Data,* 19–34.

Nargesian, Fatemeh, Ken Q Pu, Erkang Zhu, Bahar Ghadiri Bashardoost, and Renée J Miller. 2020. "Organizing data lakes for navigation." In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data,* 1939–1950.

Nargesian, Fatemeh, Erkang Zhu, Ken Q Pu, and Renée J Miller. 2018. "Table union search on open data." *Proceedings of the VLDB Endowment* 11 (7): 813–825.

Neutatz, Felix, Mohammad Mahdavi, and Ziawasch Abedjan. 2019. "ED2: A case for active learning in error detection." In *Proceedings of the 28th ACM international conference on information and knowledge management,* 2249–2252.

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. 2013. "On the difficulty of training recurrent neural networks." In *International conference on machine learning,* 1310–1318. PMLR.

Radford, Alec, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. "Improving language understanding with unsupervised learning."

Rahm, Erhard, and Philip A Bernstein. 2006. "An online bibliography on schema evolution." *ACM Sigmod Record* 35 (4): 30–31.

Ramos, Juan, et al. 2003. "Using tf-idf to determine word relevance in document queries." In *Proceedings of the first instructional conference on machine learning,* 242:29–48. 1. Citeseer.

Rekatsinas, Theodoros, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. "Holoclean: Holistic data repairs with probabilistic inference." *arXiv preprint arXiv:1702.00820.*

Sato, Motoki, Jun Suzuki, Hiroyuki Shindo, and Yuji Matsumoto. 2018. "Interpretable adversarial perturbation in input embedding space for text." *arXiv preprint arXiv:1805.02917.*

Scherzinger, Stefanie, Meike Klettke, and Uta Störl. 2013. "Managing schema evolution in nosql data stores." *arXiv preprint arXiv:1308.0514.*

Scherzinger, Stefanie, and Sebastian Sidortschuck. 2020. "An empirical study on the design and evolution of NoSQL database schemas." In *International Conference on Conceptual Modeling,* 441–455. Springer.

Shen, Yanyan, Kaushik Chakrabarti, Surajit Chaudhuri, Bolin Ding, and Lev Novik. 2014. "Discovering Queries Based on Example Tuples." In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data,* 493–504. SIGMOD '14. Snowbird, Utah, USA: Association for Computing Machinery. https://doi.org/10.1145/2588555.2593664.

Sheng, Yangjun. 2019. "Non-blocking Lazy Schema Changes in Multi-Version Database Management Systems." PhD diss., Carnegie Mellon University Pittsburgh, PA.

Sjøberg, Dag. 1993. "Quantifying schema evolution." *Information and Software Technology* 35 (1): 35–44.

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. "Dropout: a simple way to prevent neural networks from overfitting." *The journal of machine learning research* 15 (1): 1929–1958.

Stojanovic, Ljiljana. 2004. "Methods and tools for ontology evolution."

Stonebraker, Michael, Ihab F Ilyas, et al. 2018. "Data Integration: The Current Status and the Way Forward." *IEEE Data Eng. Bull.* 41 (2): 3–9.

Störl, Uta, Meike Klettke, and Stefanie Scherzinger. 2020. "NoSQL Schema Evolution and Data Migration: State-of-the-Art and Opportunities." In *EDBT,* 655–658.

Thirumuruganathan, Saravanan, Shameem A Puthiya Parambath, Mourad Ouzzani, Nan Tang, and Shafiq Joty. 2018. "Reuse and adaptation for entity resolution through transfer learning." *arXiv preprint arXiv:1809.11084.*

Van Aken, Dana, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. 2017. "Automatic database management system tuning through large-scale machine learning." In *Proceedings of the 2017 ACM International Conference on Management of Data,* 1009–1024.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. "Attention is all you need." In *Advances in neural information processing systems,* 5998–6008.

Velegrakis, Yannis, Renée J Miller, and Lucian Popa. 2004. "Preserving mapping consistency under schema changes." *The VLDB Journal* 13 (3): 274–293.

Wang, Lanjun, Shuo Zhang, Juwei Shi, Limei Jiao, Oktie Hassanzadeh, Jia Zou, and Chen Wangz. 2015. "Schema management for document stores." *Proceedings of the VLDB Endowment* 8 (9): 922–933.

Wei, Jason, and Kai Zou. 2019. "Eda: Easy data augmentation techniques for boosting performance on text classification tasks." *arXiv preprint arXiv:1901.11196.*

Wu, Renzhi, Prem Sakala, Peng Li, Xu Chu, and Yeye He. 2021. "Demonstration of Panda: A Weakly Supervised Entity Matching System." *arXiv preprint arXiv:2106.10821.*

Wu, Yonghui, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. "Google's neural machine translation system: Bridging the gap between human and machine translation." *arXiv preprint arXiv:1609.08144.*

Yu, Cong, and Lucian Popa. 2005. "Semantic adaptation of schema mappings when schemas evolve." In *Proceedings of the 31st international conference on Very large data bases,* 1006–1017.

Zhang, Ji, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, et al. 2019. "An end-to-end automatic cloud database tuning system using deep reinforcement learning." In *Proceedings of the 2019 International Conference on Management of Data,* 415–432.

Zhang, Xiang, Junbo Zhao, and Yann LeCun. 2015. "Character-level convolutional networks for text classification." *Advances in neural information processing systems* 28:649–657.

Zhao, Chen, and Yeye He. 2019. "Auto-em: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning." In *The World Wide Web Conference,* 2413–2424.

Zhu, Erkang, Dong Deng, Fatemeh Nargesian, and Renée J Miller. 2019. "Josie: Overlap set similarity search for finding joinable tables in data lakes." In *Proceedings of the 2019 International Conference on Management of Data,* 847–864.

Zhu, Erkang, Fatemeh Nargesian, Ken Q Pu, and Renée J Miller. 2016. "LSH ensemble: Internet-scale domain search." *arXiv preprint arXiv:1603.07410.*

Zou, Jia, Pratik Barhate, Amitabh Das, Arun Iyengar, Binhang Yuan, Dimitrije Jankov, and Chris Jermaine. 2020. "Lachesis: Automated Generation of Persistent Partitionings for Big Data Applications."