Data Ambiguity Profiling for the Generation of Training Examples

Enzo Veltri
University of Basilicata, Italy
enzo.veltri@unibas.it

Gilbert Badaro
EURECOM, France
gilbert.badaro@eurecom.fr

Mohammed Saeed
EURECOM, France
mohammed.saeed@eurecom.fr

Paolo Papotti
EURECOM, France
paolo.papotti@eurecom.fr

***Abstract*—Several applications, such as text-to-SQL and computational fact checking, exploit the relationship between relational data and natural language text. However, state of the art solutions simply fail in managing "data-ambiguity", i.e., the case when there are multiple interpretations of the relationship between text and data. Given the ambiguity in language, text can be mapped to different subsets of data, but existing training corpora only have examples in which every sentence/question is annotated precisely w.r.t. the relation. This unrealistic assumption leaves the target applications unable to handle ambiguous cases. To tackle this problem, we present an end-to-end solution that, given a table D, generates examples that consist of text, annotated with its data evidence, with factual ambiguities w.r.t. D. We formulate the problem of profiling relational tables to identify row and attribute data ambiguity. For the latter, we propose a deep learning method that identifies every pair of data ambiguous attributes and a label that describes both columns. Such metadata is then used to generate examples with data ambiguities for any input table. To enable scalability, we finally introduce a SQL approach that can generate millions of examples in seconds. We show the high accuracy of our solution in profiling relational tables and report on how our automatically generated examples lead to drastic quality improvements in two fact-checking applications, including a website with thousands of users, and in a text-to-SQL system.**

I. Introduction

Ambiguity is common in natural language in many forms [1]. We focus on the ambiguity of a factual sentence or question w.r.t. the data available in a table. A simple example is the question "Is Curry the best shooter in NBA history?". Based on the NBA official data, the answer changes depending on the interpretation of *shooting* in terms of table attributes [2]. Indeed, in the context of querying relational databases using natural language (text-to-SQL), "ambiguity of natural language queries is one of the most difficult challenges" [3]. The problem of data ambiguities in text is also relevant for many natural language processing (NLP) applications that use relational data. These span from computational fact checking, i.e., verify if a given claim holds w.r.t. a table [4], [5], [6], [7], to question answering in general [8], [9]. Existing solutions are trained on corpora of short text examples (sentences or questions) annotated w.r.t. the data. However, such corpora do not contain examples with data ambiguities and the methods simply fail when they are tested on such cases. This problem is important in practice. In the log of an online application for fact checking (https://coronacheck.eurecom.fr), 88% of the claims verified by the users are ambiguous sentences w.r.t. the available data.

Consider a fact checking application that verifies a textual claim, such as "Carter LA has higher shooting than Smith SF",

	Player	Team	FG%	3FG%	fouls	apps
t_1	<u>Carter</u>	<u>LA</u>	<u>56</u>	<u>47</u>	4	5
t_2	<u>Smith</u>	<u>SF</u>	<u>55</u>	<u>50</u>	4	7
t_3	*Carter*	SF	60	51	*3*	3

TABLE I. A data-ambiguous example contains the sentence "Carter LA has higher shooting than Smith SF" and the evidence underlined. Another example contains the question "Did Carter commit 3 fouls?" and the evidence in italic.

against a relational table D as in Table I. Even as humans, it is hard to state if the sentence is true or false w.r.t. the data in D. The challenge is due to the two different meanings that can be matched to *shooting*: the claim can refer to attribute *Field Goal* (FG%) or to *3-point Field Goal* (3FG%). The same challenge applies with a SQL query expressed in natural language such as "Did Carter commit 3 fouls?". We refer to this issue as *data ambiguity*, i.e., the existence of more than one interpretation of a text w.r.t. the data for a human reader.

While existing corpora of examples come from extensive and expensive manual efforts, they do not contain examples with ambiguous text. Existing applications fail in these scenarios: the two examples above would lead to a single interpretation, which is incorrect in 50% of the cases. Since manually crafting examples is not feasible at scale, we propose to generate examples with data ambiguities from the data itself.

Consider again the first example about the higher shooting percentage. If we know what the ambiguous attributes are, the data relevant for such example is one of the result tuples obtained from a SQL query $q1$ executed over D comparing the **FG%** and **3FG%** attributes for pairs of players.

```
q1: SELECT b1.Player, b1.Team, b2.Player,
           b2.Team, b1.FG%, b2.FG%,
           b1.3FG%, b2.3FG%
    FROM D b1, D b2
    WHERE b1.Player <> b2.Player AND
          b1.Team <> b2.Team AND
          b1.FG% > b2.FG% AND
          b1.3FG% < b2.3FG%
```

Once we have identified these cells from D, we can create the corresponding text by using a data-to-text generation method [10], [11]. These methods, given a table and a set of table cells, produce their textual description. A generated example therefore consists of the SQL query, the text, and the cells, or evidence, that address the text in the test.

In this paper, we describe PYTHIA[1], an end-to-end system

[1]PYTHIA was the name of the Oracle of Delphi, a priestess famous for her enigmatic prophecies that needed interpretation.

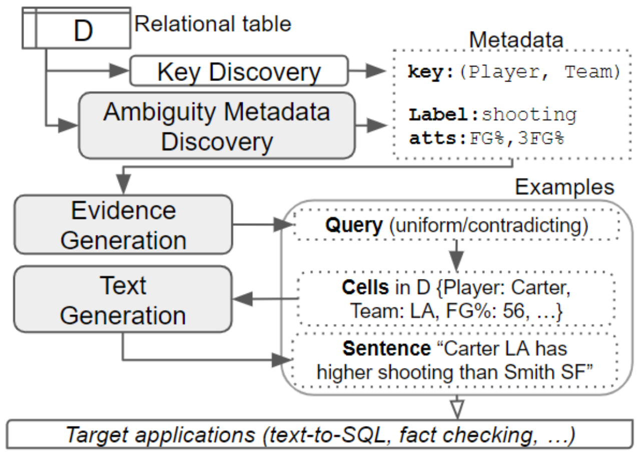
D
Relational table
Key Discovery
Metadata
key:(Player, Team)
Ambiguity Metadata Discovery
Label:shooting
atts:FG%,3FG%
Examples
Evidence Generation
Query (uniform/contradicting)
Text Generation
Cells in D {Player: Carter, Team: LA, FG%: 56, ...}
Sentence "Carter LA has higher shooting than Smith SF"
Target applications (text-to-SQL, fact checking, ...)

Fig. 1. PYTHIA takes as input a table and computes traditional and new metadata for it. The ambiguity discovery identifies pairs of attributes and a label that generalizes them. The metadata is used to produce the query that identifies the evidence cell values that lead to an ambiguity. Such cells are then processed to generate a short text. Examples (query, table cells, sentence) are finally used in target applications, e.g., as training data.

to generate examples that are data ambiguous w.r.t. a given table [12]. Given a relation, the key idea is to identify the set of cells involved in ambiguous attributes with a query over the data. These cells are then used to generate the text. However, coming up with the query automatically from a given relational table depends on the ability of identifying and characterizing the ambiguities. Consider again query $q1$ above. First, the ambiguous attribute pair is not given as input, e.g., FG% and 3FG% in *q1* must be identified. Second, the data-to-text generators are not designed to create sentences with ambiguity, but these are the examples that we aim at creating. In order to extend such solutions, they need some extra information in their prompt. We extend the prompt using a *label* that plausibly refers to the two attributes, e.g., "shooting" in *q1* is a label for FG% and 3FG%. To handle these challenges, we introduce deep learning models that predict this new kind of *ambiguity metadata* from the table, as depicted in Figure 1.

In this work, we show that relational data can play an important role in the *data-centric AI* movement [13], which argues that more attention should be put on the role of data, rather than focusing only on the creation of more complex ML models. We tackle the problem of generating examples with data ambiguity from relational data and build PYTHIA around the following contributions:

1) We formalize the problem of generating text that is ambiguous w.r.t. a dataset and characterize its properties. We model the problem as the generation of data evidence (with SQL queries) and generation of text (Section II).
2) Evidence and text generation require metadata about ambiguity over the attributes, i.e., pairs of ambiguous attributes and their corresponding label, such as attributes FG% and 3FG% with 'shooting'. We introduce a solution based on transfer learning and a transformer architecture for the inference of such metadata from the given table. As the problem is new, we use weak supervision on 500k tables to create training data for such task (Section III).
3) We design a generic solution that enables the generation of examples with data ambiguity. The algorithm is instantiated with the metadata for the dataset at hand to produce the (query, data evidence, text) triples (Section IV).
4) We focus on genuine attribute ambiguity, in contrast to other ambiguity cases where the meaning is entirely clear to a human but an algorithm detects more than one alternative[2]. We therefore conduct a user study with 10 human annotators to create the first test dataset of 252 ambiguous attribute pairs and corresponding labels for a set of 13 relational tables (Section V).
5) PYTHIA creates thousands of examples, with an f-measure of 87% and 82% in the prediction of pairs and labels, respectively. Examples are correctly recognized as data ambiguous by human judges. Generated, unverified examples are used for training the first fact checking and text-to-SQL applications that handle data ambiguity (Section VI).

Related work is discussed in Section VII and possible future research directions are presented in Section VIII. Code and datasets are available online [14].

II. Overview and Problem Formulation

An example is a triple composed of a text that is data ambiguous w.r.t. a table D, the subset of D's cells that are needed to verify or answer such text, and the query that identifies such subset. Examples can be used in training applications to make them aware of the ambiguity problem. Figure 1 shows the high-level architecture of PYTHIA. Our approach starts by deriving the ambiguity metadata for a given table. With such metadata, it then creates SQL queries that identify all the subsets of cells involving ambiguity. The results of these queries over the table are then used to complete the example by creating the corresponding natural language text.

We now introduce the definition of data ambiguous text, the queries that use ambiguity metadata to identify the subset of data that lead to the examples, and how text can be generated from data. We then conclude with the problem formulation.

A. Text with Data Ambiguity

Consider a short natural language text, either sentence or question, s and a relational table t. A text s is *factual* if it contains only information (entities, values) that are verifiable/answerable with the relevant cells in t, namely *evidence*. We define a factual text s as *data ambiguous* if a human judges that it has more than one possible meaning w.r.t. schema and content in table t. We refer to the triple (query, data evidence, text) as an *example* for the target applications[3].

Ambiguity Structure. Text can be data ambiguous in different ways. For example, s_1: "Carter LA has better shooting than Smith SF" is ambiguous w.r.t. relation D in Table I because even as humans we cannot state for sure if it refers to regular Field Point statistics or performance for the 3 point range. This is an example of *attribute ambiguity* over two attributes, but a text could be ambiguous up to a number of attributes equal to the arity of the table.

Factual question s_2: "Did Carter commit 3 fouls?" shows a different kind of data ambiguity w.r.t. D. As rows are identified

[2]For example, for a human "Carter SF has played 3 times" clearly refers to player *Appearances* (apps), but an algorithm could match both fouls and apps.

[3]Not all target applications necessarily use the three elements, e.g., in text-to-SQL the evidence is not used by most systems.

Attribute, Contradictory:
"Carter LA has better shooting than Smith SF"
Structure: $\{t_1 - t_2\}$ rows, {FG%, 3FG%} attributes
Match: true, false

Row, Uniform: "Carter has more appearances than Smith"
Structure: $\{t_1 - t_2,\ t_3 - t_2\}$ rows, {apps} attribute
Match: false, false

Full, Contradictory: "Carter has better shooting than Smith"
Structure: $\{t_1 - t_2,\ t_3 - t_2\}$ rows, {FG%, 3FG%} attributes
Match: true, false, true, true

Fig. 2. Example of different ambiguous text w.r.t. D.

by two attributes in this table, a reference to part of its key (Player) makes it impossible to identify the right player. This is a case of *row ambiguity* over two records, but a text could be ambiguous to a number of rows up to the size of the table.

A text can be ambiguous in the two dimensions, both over the attributes and the rows, and the number of possible interpretations increases accordingly. We therefore classify the structure of the ambiguity in *attribute*, *row*, and *full*. Depending on the structure, a factual text can span multiple attributes and multiple rows, e.g., s_1 involves two rows and three attributes (two ambiguous), while $s2$ involves one row and two attributes.

Ambiguity Match Type. We further distinguish between *contradictory* and *uniform* texts. The first type has interpretations with opposite factual match w.r.t. the table. Text s_1 and s_2 are both contradictory: one interpretation is confirmed from the table and one is refuted. For uniform text, all interpretations have equal factual matching w.r.t. the table, i.e., alternative interpretations are either all true or all false. For example, "Carter has more appearances than Smith" is false for both Carter players in D. We found the distinction between *contradictory* and *uniform* text crucial in target applications as they give control over the generation of the example corpora.

We summarize the different kinds of text in Figure 2. As shown in its last example, text can be ambiguous in more than one structure at the same time (full), but it cannot be at the same time uniform and contradictory. The full ambiguity leads to four interpretations for the last text. We list them pivoting on attribute ambiguity first, i.e., considering FG% and $t_1 - t_2$, FG% and $t_3 - t_2$, 3FG% and $t_1 - t_2$, 3FG% and $t_3 - t_2$.

In the following, we focus on examples in the form of factual sentences, such as s_1, but the generation of ambiguous text equally applies to factual questions, such as s_2.

B. Generating Evidence

An *ambiguity query* (or *a-query*) executed over table D returns the *evidence* for the example, i.e., the set of cells that define an ambiguity w.r.t. D. Every result tuple corresponds to the evidence for one example. For instance, consider again *q1* from Section I. It returns all the pairs of distinct players that, depending on the attribute interpretation, lead to contradictory sentences. In every a-query, we collect the evidence by listing in the `SELECT` clause the distinct attributes in the query.

Attribute Ambiguity. A set of cells forms evidence for attribute ambiguity if at least one value comparison involve two or more ambiguous attributes. Consider again a-query *q1* for attribute ambiguity. The a-query is a self-join on the composite key (no row ambiguity) and the selection conditions are defined on the ambiguous attributes (e.g., the different FGs%). Assuming we can obtain with profiling the name of the relation and the names of the attributes in the keys, the main challenge is to identify the ambiguous attributes (FG% and 3FG% in the example).

Row Ambiguity. A set of cells forms evidence for row ambiguity if the selection conditions do not include any complete set of key attributes. One a-query to produce row ambiguity evidence exploits the key metadata, specifically the presence of composite keys in the table. In this case, the query selects evidence using only one of the attributes in its set. A-query *q2* produces evidence for ambiguous text such as s_2.

```
q2: SELECT b1.Player, b1.Fouls
    FROM D b1, D b2
    WHERE b1.Player = b2.Player AND
    b1.Fouls <> b2.Fouls
```

Query Properties. A-queries also set the match type of the generated examples. In a-query *q1*, the last two clauses enforce that the examples are contradictory, but it can be modified to obtain uniform examples by changing one of their operators, e.g., change '<' to '>' in the last clause over attribute 3FG%. By removing the last two clauses in *q1*, it returns both contradictory and uniform sentences. The same reasoning applies for row ambiguity. A-query *q2* identifies contradictory examples, but it returns uniform examples by changing the last atom from '<>' to '='. As for $q1$, uniform and contradicting examples are obtained by removing the last predicate.

As datasets can evolve over time, a-queries can be efficiently re-run to create updated examples.

C. Generating Text From Data

Data-to-text is the task of generating natural language text from structured data, specifically relational data in our case [11], [15], [10]. Given a table and a set of table cells, the goal is to produce a one-sentence description. These methods rely on pre-trained language models, such as T5, which are fine tuned on corpora of pairs of examples (text, evidence) for the data to text generation. The examples consist of manually crafted sentences that involve reasoning and comparisons among rows, columns, or cells. For example, given the evidence for a table about sports events, with the name of an athlete ("Becker") in the title, and four cells with year ("1995"), competition ("World Championships") and events ("100m", "4x100 m relay"), the corpus contains the sentence "Becker competed at the 1995 World Championships both individually and in the relay".

The existing corpora for this task are large and manually curated, but ambiguity examples are not provided in the existing resources. However, we show that it is possible to extend the original prompt to generate data ambiguous text from the given evidence. For attribute ambiguity, it is crucial to inform the prompt with a label that abstracts the two ambiguous attributes. Using the ambiguity metadata in the existing data-to-text models, we generate high-quality text for our examples.

D. Problem Definition

Given a triple (t, st, mt) with a relational table t, the structure type st, and the match type mt, we want to obtain all

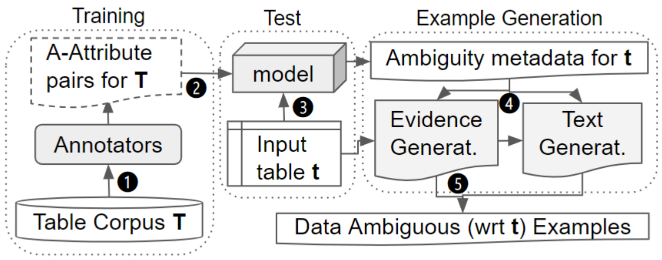

Fig. 3. Overview of the solution for ambiguity metadata discovery. In the training phase, a model is fine-tuned for the prediction task. Given a table t, the model generates the metadata that is then used to generate examples.

the example triples (query, evidence, text) that satisfy st and mt. As our solution relies on the a-queries, to generate evidence, and the data-to-text model, to obtain sentences/questions, the challenge is how to obtain the metadata to support such tasks. Specifically, given the table t, we must obtain the *pairs of attributes that raise ambiguity*, for the evidence generation, and their *label*, for the text generation.

III. Predicting Ambiguity Metadata

From our analysis of an online fact checking application's log, we estimate that 40% of the user submitted sentences present attribute ambiguity. Information about keys, for queries such as *q2*, is automatically obtained with any of the existing data profiling methods [16]. We therefore focus our effort on the discovery of ambiguity metadata, which is needed for evidence and text generation for queries such as *q1*.

A. Ambiguity Metadata Discovery

We describe the overview of the solution (Figure 3) to obtain the ambiguity metadata for a given relational table.

❶ In the training phase, a transformer pre-trained model is fine-tuned with examples of pairs of ambiguous attributes[4]. As training data does not exist for this task, we use six unsupervised, noisy annotator functions. Given a pair of attributes for a table, the annotators produce a label for them if they are ambiguous. For example, given the attributes 'length' and 'width', an annotator function should produce labels such as 'dimension' or 'magnitude' that yield some ambiguity w.r.t. the input attributes. We run these noisy annotators on a large collection of tables, such as the WebTables corpus [17]. The unsupervised functions are detailed in Section III-B.

❷ The noisy output of the annotator functions is used as training examples for a fine-tuning task on an encoder-decoder language model [18]. This text-to-text architecture achieves state-of-the-art results on several NLP tasks. We introduce the novel task of predicting (1) if two attributes are ambiguous and (2) the label for such ambiguity. We design two variants that take into consideration different levels of access to the tables. One considers as input the schema only, while the second assumes that a sample of the data can be fed to the model. We discuss this component in Section III-C.

[4]We focus on pair of attributes as we found it effective for example generation in our target application. The method can be extended with a post-processing step that analyzes pairs to compute larger sets, e.g., 1pt%, 2pt%, 3pt%.

TABLE II. Examples of the aliases for two input attributes from five annotators: synonyms (Syn), related to (RelTo), derived (der), subtypes (isA), and Wikipedia results (Wiki).

Input	syn	relTo	der	isA	Wiki
silver medal	silver	runner up	medal	trophy	medal
earnings	wage	profit	earn	giving	income

❸ Once the model has been trained, we use it at test time with any unseen input table t. We test all the possible pairs of non-key attributes in t and, if a label is predicted, we consider the input attribute pair in question as ambiguous. We model the output attribute pair and their label as metadata for t.

❹ For every attribute pair, we instantiate the evidence generation module and execute it over t. Every set of cells from the evidence generation is passed to the text generation, together with the label for the current attribute pair and the table name.

❺ Once the data evidence and the text are generated, they form the examples that are data ambiguous w.r.t. table t.

B. Annotator Functions

Our goal is to create training data for a model that, given two attributes alongside table schema (or table schema with a data sample), either produces a label if both attributes are ambiguous or abstains otherwise. Unfortunately, we cannot count on a large amount of manually annotated examples. To obtain such data in an unsupervised fashion, we start the process with a set of basic annotator functions that exploit external resources and heuristics to find candidates for ambiguity.

The first set of our weak supervision annotators [19] are designed in a two step approach. First, an *alias function* automatically finds possible *aliases* for a given word. For example, a set of possible aliases for the word length are {measurement, measure, range}. Then the alias for two candidate attributes are compared and eventually selected. We design alias functions that use different external datasets. Four alias functions use the ConceptNet graph [20]. Given the input word (attribute label), we look for its relationships in the graph that are alternative representations of the input: $a)$ syn synonyms; $b)$ $relTo$ related words; $c)$ der words it derived from; and $d)$ isA subtypes. For a second external resource, we use the Wikipedia API to search top page titles as other possible aliases with the *wiki* function. Table II shows examples of the generated aliases.

Once the aliases have been collected, given a pair of attributes represented by their names, we say that two attributes are ambiguous if the intersection of their aliases values is non-empty. The ambiguous labels are the intersection. If the intersection is empty, then the two pairs of attributes are not ambiguous. An annotation function makes use of the intersection for every external resource. If the attribute names are not meaningful, the annotator functions output empty results, e.g., attribute with name "A12" has empty results for all annotator functions.

As a sixth function, we find the Longest Common Substring (*LCS*) between two attribute names. Since the result may contain sequences of characters without meaning, we filter the generated word with a dictionary.

C. Fine-Tuning the Language Model

Pre-trained transformer based language models (LMs) such as BERT [21], RoBERTa [22], XLNet [23] and T5 [18] have shown to perform well in different NLP tasks such as question answering and text classification. Typically, these models are pre-trained on large text corpora such as articles from Wikipedia, news articles, or Common Crawl. The model is pre-trained in an unsupervised manner, for example by predicting a missing token or the next-sentence in a paragraph. Unlike conventional word embedding techniques such as Word2vec [24], GloVe [25], or FastText [26], pre-trained transformer-based LMs learn better the semantics of words and provide different representations for the same word when utilized in different contexts. LMs can then be further *fine-tuned* for supervised, specific tasks. Tuning with task specific data is one of the advantages of pre-trained LMs, thanks to the advances in transfer learning. Fine-tuning is performed in a supervised manner by providing the LM the labeled input. Fine-tuning allows to generate tasks with new prompts while effectively exploiting transfer-learning for natural language understanding.

We define two variants of the same fine-tuning task with the objective of identifying a pair of ambiguous attributes and their common label, in case such label exists. We consider a sequence generation task where the goal is to learn a function $f : x \rightarrow y$, where x is a sequence of text containing the pair of attributes and y is a sequence of text corresponding to the label (where *none* means no ambiguity). The training data comes from the annotator functions discussed above. The difference between the two tasks lies in the prompt provided to the LM.

In the *schema-task*, function f takes a table schema T_s and a pair of its attribute with labels P_a as input, the goal is to predict a text label l in case a similarity exists between the two attributes or *None*. Hence $x = (T_s, P_a); y \in (l, None)$.

In the *data-task*, the table header and randomly selected rows are concatenated with the attributes along with the corresponding label. In this case, f' assumes as input a sample of the table data cells T_d, in addition to T_s and P_a. We denote the schema and the data together as T_{sd}. Hence, $x' = (T_{sd}, P_a); y \in (l, None)$. For this task, we discuss alternative representations of the table cells, namely a row serialization and a column serialization. Our decision of the two alternatives is based on the typical serialization in the literature to create neural representations for database tables [27], [28], [29], [30], [31]. While other contextual features are sometimes encoded, we keep our model simple and generic.

Figure 4 shows a sample of the prompt for the fine tuning of the model based on the basket data in Table I. The special tokens are used to help the model distinguish the start and the end of a cell, of a row, and of a column depending on the configuration.

Schema-task prompt: concatenate T_s, P_a, l
Player | Team | FG% | 3FG% | fouls | apps, attr1: FG% attr2: 3FG%, [y: shooting]

Data-task prompt (**Row**): concatenate T_{sd}, P_a, l
Player | Team | FG% | 3FG% | fouls | apps || Carter | LA | 56 |47 | 4 | 5 || Smith | SF | ... | 7 || Carter| ..., attr1: FG% attr2: 3FG%, [y: shooting]

Data-task prompt (**Column**): concatenate T_{sd}, P_a, l
Player | Carter | Smith | Carter || Team | LA | SF | SF || FG%| 56 | 55 | 60 || 3FG% | ... || fouls | 4| ..., attr1: FG% attr2: 3FG%, [y: field goal]

Fig. 4. Examples of input prompts for fine-tuning T5 model for the two tasks. "|" represents a cell separator token and "||" represents row and column separators tokens.

IV. Example Generation

We now describe how to go from a given table t, with its key and ambiguity metadata, to the examples for the target applications. Our default approach starts with the generation of the data evidence, which is then used to create the corresponding text. We also present an alternative approach, which enables scalability but requires human defined prompts.

A. Evidence and Text Generation

Given a table t and its ambiguity metadata, we present an algorithm to generate a-queries, obtain the evidence, and finally generate the text for every example. The algorithm assumes the availability of the auxiliary functions presented in Section III. Moreover, for a table t, it gets its composite keys with $t.CK$ and the primary key with $t.PK$; we employ profiling methods to obtain metadata about the keys [16], if such information is not available in the input table. We assume that the attributes in a CK are ordered by the number of unique values for each attribute in an increasing order. Auxiliary functions $sub(ck)$ and $last(ck)$, with $ck \in t.CK$, exploit this assumption and return all the attributes except the last, and the last attribute, respectively. If no primary key is present, we add a synthetic one. Function $t.AmbAttributes$ returns a triple $\langle amb1, amb2, label \rangle$, where $amb1$ and $amb2$ are the names of the ambiguous attributes and $label$ is the ambiguous word. Finally, function *genText* generates sentences or questions in natural language through a generator that takes as input the evidence. Depending on the ambiguity type, it makes use of the ambiguous label.

Algorithm 1 generates examples given a table t, the desired match type *mt*, the operators to use *ops*, and a text generator module gen. The match types are contradictory and uniform. Possible comparison operators for numerical attributes are '$<$', '$>$', '$=$' and '$<>$', while only '$=$' and '$<>$' are for attributes with categorical values. We model the text generator as an input to stress that the solution can benefit of the improvements coming from new data-to-text modules in the ML community. Our default text generator is a T5 pre-trained model fine-tuned on the ToTTo data-to-text dataset [10]. The algorithm can be easily modified to take the ambiguity structure as input as well.

The process starts by iterating, for each operator and each match type, over the three possible types of ambiguity (lines 2-9). We get the negation of op as nop, if the match type is "uniform" then nop and op should be equal (lines 5-6). It then calls the functions that return the examples (lines 7-9) and the union of those are returned (line 10).

For every type of ambiguity, a function gets as input the table, the match type and the operator (lines 11, 18, 26). Depending on the ambiguity type, different auxiliary functions are used. For attribute ambiguity (lines 11-17), the pairs of ambiguous attributes and the label are obtained (line 13). The

Algorithm 1: Generate Examples

```
   Input: table $t$; match type $mts$; operators $ops$; text
          generator $gen$;
   Output: Set of (query, evidence, text) examples $exs$
1  $exs = \{\}$
2  foreach operator $op \in ops$ do
3      foreach match type $mt \in mts$ do
4          $nop = neg(op)$
5          if $mt$ == 'uniform' then
6              $nop = op$
7          $exs$ += $attrAmb(t, op, nop)$
8          $exs$ += $rowAmb(t, op, nop)$
9          $exs$ += $fullAmb(t, op, nop)$
10 return $exs$ //return examples
11 Function attrAmb(t, op, nop):
12     $exs = \{\}$; $pk = t.pk$
13     $attrAmb = t.ambAttributes$ //return pairs with $label$
14     foreach attr1, attr2, label $\in$ attrAmb do
15         $q$, $evi$ = $attrData(t, pk, attr1, attr2, op, nop)$
16         $exs$ += ($q$, $evi$, $genText(evi, label, gen)$)
17     return $exs$
18 Function rowAmb(t, op, nop):
19     $exs = \{\}$; $cks = t.ck$; $pk = t.pk$
20     foreach $ck \in cks$ do
21         $attrNoCKs$ = $t.attr$ - $ck$
22         foreach $attr \in attrNoCKs$ do
23             $q$, $evi$ = $rowData(t, ck, attr, op, nop)$
24             $exs$ += ($q$, $evi$, $genText(evi, null, gen)$)
25     return $exs$
26 Function fullAmb(t, op, nop):
27     $exs = \{\}$; $cks = t.ck$; $pk = t.pk$
28     $attrAmb = t.ambAttributes$
29     foreach attr1, attr2, label $\in$ attrAmb do
30         foreach $ck \in cks$ do
31             if attr1, attr2 $\notin$ ck then
32                 $q$, $evi$ = $fullData(t, ck, attr1, attr2, op, nop)$
33                 $exs$ += ($q$, $evi$, $genText(evi, label, gen)$)
34     return $exs$
35 Function attrData(t, pk, a1, a2, op, nop):
36     $q$ = "SELECT t1.pk, t2.pk, t1.a1, t2.a2 FROM t t1, t t2
         WHERE t1.a1 op t2.a1 $\wedge$ t1.a2 nop t2.a2"
37     return $q$, $q(t)$
38 Function rowData(t, ck, attr, op, nop):
39     $sub$ = $sub(ck)$; $last$ = $last(ck)$
40     $q$ = "SELECT t1.attr, t2.attr, t3.attr, t1.ckAtts, t2.ckAtts,
         t3.ckAtts FROM t t1, t t2, t t3 WHERE t1.sub = t2.sub $\wedge$
         t1.last $\neq$ t2.last $\wedge$ t1.attr op t3.attr $\wedge$ t2.attr nop t3.attr"
41     return $q$, $q(t)$
42 Function fullData(t, ck, a1, a2, op, nop, mt):
43     $sub$ = $sub(ck)$; $last$ = $last(ck)$; $bt$=t2
44     if $mt$ == 'contradicting' then
45         $bt$ = t1
46     $q$ = "SELECT t1.a1, t2.a1, t3.a1, t1.a2, t2.a2, t3.a2,
         t1.ckAtts, t2.ckAtts, t3.ckAtts FROM t t1, t t2, t t3
         WHERE t1.sub=t2.sub $\wedge$ t1.last$\neq$t2.last $\wedge$ t1.a1 op t2.a1
         $\wedge$ t1.a2 nop t2.a2 $\wedge$ bt.a1 op b3.a1 $\wedge$ bt.a2 nop b2.a2"
47     return $q$, $q(t)$
```

former is used in the evidence generation (line 15) and the latter in the text generation (line 16). Similarly, for the row and the full ambiguity, the required metadata is collected and the respective functions to collect the evidence are executed (lines 23 and 32, respectively). We only consider pairs of ambiguous attributes with the same type (numerical or categorical).

Data evidence and label:
$t1.pk$ = (Player:Carter; Team:LA), $t2.pk$ = (Player:Smith, Team:SF), $t1.a1$ = FG%:56, $t2.a1$ = FG%:55, $t1.a2$ = 3FG%:47, $t2.a2$ = 3FG%:50, *label* = "shooting"
Text generator input: (linearized data)
(1) Player:Carter — Team:LA — shooting:56
…
(4) Player:Smith — Team:SF — shooting:50

Output text:
(1) "Carter from LA has a shooting of 56"
…
(4) "Smith in SF team had 50 as shooting accuracy"

Fig. 5. Examples of the input prompt for the text generation task. "—" is a cell separator token. ":" is the separator between cell value and attribute name.

The evidence generation is handled with one function for ambiguity type (lines 35, 38, 42). For this task, we rely on queries that only need to be instantiated for the schema at hand, following the sketch-based slot-filling approach [3]. Every variable is filled by a parameter passed to the function, e.g., for the attribute ambiguity, the pair of ambiguous attributes are passed as $a1$ and $a2$ from the previous function (lines 35-36). For the sake of the space, we use *ckAtts* and *sub* as placeholders in queries for all the attributes in a composite key ck or the attributes returned by $sub(ck)$. (lines 40 and 46). Once generated, the query is executed over t and passed as results (lines 37, 41 and 47).

After the data evidence has been collected, this is processed for the text generation to produce the complete examples. Evidence is processed together with the label for the cases involving attribute ambiguity (lines 16 and 33). In practice, we fine-tune a T5 model using a simple text linearization of the cells [32], but instead of using the attribute names of the ambiguous cells, we use their respective labels. An example of text generation for attribute ambiguity is shown in Figure 5.

Examples. To show the role of the operators, consider query $q2$ and '>' instead of '<>'. The algorithm generates a new a-query $q2'$ that creates evidence for text such as 'Carter has more than 3 fouls', we report in violet the schema-specific parts of the query (lines 22 and 40).

```
q2': SELECT b1.Player, b1.Fouls, b2.Fouls
    FROM D b1, D b2
    WHERE b1.Player=b2.Player AND
          b1.Fouls > b2.Fouls
```

To show the generality of the approach, consider a different table about COVID-19 data with schema *Covid (country, date, vaccinated, positive)* and again row ambiguity with operator '='. An ambiguous sentence for this table is "Italy has 45'900'000 persons vaccinated.", as *country* and *date* form a composite key. The resulting a-query is

```
q2'': SELECT b1.Country, b1.Vaccinated,
             b2.Vaccinated
    FROM Covid b1, Covid b2
    WHERE b1.Country = b2.Country AND
          b1.Vaccinated <> b2.Vaccinated
```

B. Ambiguity Templates For Fast Generation

As the data-to-text operation is not very scalable, we also present an alternative approach to the example generation. The overall pipeline is the same with the important difference that the query, obtained in the evidence generation, creates also the text directly in the Select clause. To generate the example with data and text, we now show how to consume all ambiguity metadata in the evidence generation step, thus relying only on query execution without external text generator modules. For this goal, we use query templates, where the intuition is to simplify the general task of full-fledged generation by predicting only certain parts of the a-query.

Definition. Let $Q : (P)$ denote a parameterized query template, where P are the parameters, with dimensionality equal to the number of parameters in Q, and R is the set of possible queries instantiated from Q over the given database D. A data ambiguous text is the non-empty query result of query $r \in R$ when executed over D.

Parameters. The parameters identify the elements of the query and can be assigned to (i) relation and attribute names, (ii) a set of textual labels, and (iii) comparison operators.

Query Properties. Parameters (i) and keys are required in all templates, while the requirements of labels and operations depends on the template and on the input match type. Indeed, templates can be instantiated in different ways to satisfy the different properties (attribute/row/full, contradicting/uniform) in the generated examples.

Templates Types. The three kinds of ambiguity have been identified from the analysis of an online fact checking application with a log of 20K sentences submitted by users. The three templates cover the vast majority of ambiguity claims that have been submitted by users and have proven to be effective in other target applications. More templates can be crafted to model complex sentences and the limit is SQL expressive power, as we discuss at the end of this section.

Attribute Ambiguity. Consider the following example of a template for contradictory attribute ambiguity; parameters are highlighted in italic.

```
Q1: SELECT CONCAT(b1.PK, print(Op,label1),
           b2.PK)
    FROM relation b1, relation b2
    WHERE b1.PK <> b2.PK AND
          b1.A1 Op b2.A1 AND
          b1.A2 neg(Op) b2.A2
```

To instantiate *Q1*, we need the name of the *relation*, attribute labels for the primary key *PK*, the names of the ambiguous attributes *A1* and *A2* (e.g., the different FGs%), the word(s) that describe the two attributes *label1* (e.g., “shooting”), and the numerical operator *Op* (e.g., ‘>’).

Function *neg(operator)* returns the opposite operator and function *print(operator,label)* returns a string that combines the two according to the operator value, e.g., “has higher shooting than”. Different *print* functions can be defined, for different sentences. Ambiguity template *Q1* leads to sentences in the form subject-verb-object: *some subject* followed by *verb* (eventually with label), followed by *some object or value*. This form changes according to the template’s Select clause.

Template $Q1$ has 6 parameters, which are obtained from the analysis of the input relation. We obtain with standard profiling the name of the relation, the names of the attributes in the primary key, and the type information for all attributes (to infer the allowed operators). The remaining 3 parameters are obtained by our metadata discovery: “shooting” is assigned to *label1*, while FG% and 3FG% to *A1* and *A2*, respectively. Once all the parameters are identified, we obtain a-query *q1*.

```
q1: SELECT CONCAT(b1.Player, b1.Team,
           'has higher shooting than',
           b2.Player, b2.Team)
    FROM D b1, D b2
    WHERE b1.Player <> b2.Player AND
          b1.Team <> b2.Team AND
          b1.FG%>b2.FG% AND b1.3FG%<b2.3FG%
```

In *Q1*, the records are precisely identified by the primary key (no row ambiguity), while the label spans two attributes (attribute ambiguity). The last two conditions in the template enforce that output sentences are contradictory and are modified according to the problem input as detailed in Algorithm 1.

Row Ambiguity. The row ambiguity template exploits the presence of composite keys. The template selects rows using only one of the attributes in the set composing the key.

```
Q2:SELECT CONCAT(b1.sub(CK), print(Op),
         b1.A, A)
   FROM relation b1, relation b2
   WHERE b1.sub(CK) = b2.sub(CK) AND
         b1.A neg(Op) b2.A
```

Template $Q2$ leads to a-query $q2$ that produces data ambiguous sentences such as ‘Carter has 3 fouls’. In terms of parameters, $Q2$ does not need a label. The print function automatically derives the text from *Op* alone, e.g., “has” is assigned to ‘=’ and “has more than” to ‘>’. As the relation name is given, the template works for any non key numerical attribute (e.g., Fouls) and any operator. As for *Q1*, a modification to its `WHERE` clause changes its output from contradictory to uniform.

Full Ambiguity. The third kind of templates returns examples that are both attribute and row ambiguous.

```
Q3: SELECT CONCAT(b1.sub(CK),
           print(O1,label1), b2.sub(CK))
    FROM relation b1, relation b2
    WHERE b1.sub(CK) = b2.sub(CK)
```

Template *Q3* returns both uniform and contradicting examples.

Templates vs Text Generation. The two approaches have different strengths and limitations. The text generation approach is our default choice as it is fully automatic, i.e., it does not require users to write templates. Moreover, the text generation approach leads to variety in the generated text, while the text generated by the templates is uniform, unless multiple Select clauses are manually crafted.

On the other hand, templates are much more scalable, with millions of examples generated in seconds. While this is rarely needed in target applications, it immediately enables the generation of large corpora. Another important aspect is that templates are readily extensible. Templates $Q2$ and $Q3$ are instantiated in the presence of composite keys. However, we can generate row ambiguity sentences by using a forth template that uses the attributes involved in functional dependencies (FDs). There are also attribute ambiguous sentences, such as "SF has the worst scorer", that require an a-query using an aggregate function. The intrinsic limit of the template approach is the expressive power of SQL. For the sake of space, we report FD and aggregate templates in the extended version of this paper, however those are just variants of the three templates above [33]. The extended version also reports the algorithm to generate a-queries given a template and a table.

V. Attribute Ambiguity Dataset

To evaluate the performance of the fine-tuned models on unseen tables, we designed two annotation tasks whose goal is to manually identify whether a given label is similar to one of the table attributes. For example, given the word "shooting", we want to know what are the attributes in D (Figure I) that are similar in meaning. Two attributes that share a word are marked as ambiguous and that word is a label for such pair.

Tables. We selected 13 tables from different sources. Four datasets from the popular UCI repository [34] (Abalone, Adults, Iris and Mushroom), two versions of a Basket dataset, used in NLP text generation challenges, similar to D (one version with full name attributes and another with acronyms) [35], and 7 relational tables with headers from the WebTables corpus [17].

We provided the initial labels by filtering those from the automatic annotators (Section III-B) and by hand crafting more options. We also allowed participants to suggest other possible ambiguous labels for the schema.

Tasks: Given a table, our goal was to annotate all the ambiguities in the columns when looking at different parts of the data. We therefore asked the participants to annotate ambiguities for two tasks where (i) the *schema* or (ii) the *schema and the data* were provided.

Schema only. Given a table schema, we asked to annotate ambiguous candidate labels w.r.t. the attribute by looking only at the *schema* of the table. Participants wrote **1** if they were confident that the label has the same meaning of the corresponding attribute (e.g., a clear synonym or hypernym), **0.5** for uncertain cases, and **0** for not related.

Schema and Data. Given the schema and a sample of the data, we asked to match the labels and the attributes with the same logic of the previous task. Given that participants also have access to data in this task, it is expected that some of the attributes that are ambiguous in task 1 are no longer ambiguous when both schema and data are available, or vice versa. For example, it is clear that attribute "workclass" can be matched to label "situation" after being able to see values such as "Self-emp-inc" (incorporated self employment).

Example. Consider the UCI Adult dataset and a subset of its schema with the following attributes: age, workclass, sex, capital-gain, capital-loss, hours-per-week, native-country, and salary. The label "gender" will have a score of 1 with the "sex" attribute. The label "income" will have a score of 1 with "capital-gain" and "salary", but one may mark "capital-loss" as an uncertain match.

Annotation Process. Ten participants were selected among graduate students in a CS department. All participants have at least C2 English language proficiency (Common European Framework of Reference). Each combination *table*, *task* was assigned to three participants.

From Annotations to Ground Truth. For each attribute, we obtain a list of labels with non-zero score from the annotators. We add a label to the ground truth if it appears at least 2 times with score 1 in the list for an attribute. In case of unclear decisions, such as scores $\langle 0, 0.5, 1 \rangle$ for a label, we used an additional participant to solve the uncertainty.

We stated that two attributes are ambiguous if they share the same label. We observe that the same pair of attributes shares more than one label. For example, "occupation" and "workclass" attributes are ambiguous with possible labels {"profession", "status", "position", "work", "occupation", "employment", "job"}. In this process, we model the attribute ambiguity as a binary test, i.e., pairs with a majority of 0.5 scores are considered as ambiguous. We leave the study of different degrees of ambiguity to future work.

The final test set contains 1321 pairs of attributes. Among them, 252 pairs are marked as ambiguous, with an average of 1.8 labels.

Quality Assurance. We measured the agreement across participants with Krippendorff's Alpha-Reliability coefficient [36] and obtained 0.452 and 0.305 for the schema without and with data, respectively. As a reference, 0.6 is a reasonable coefficient for the task of labeling sentiments in English tweets [37]. We explain the low scores for our tasks with two arguments. First, our task of judging ambiguous labels has higher complexity compared to the task of labeling sentiment in short texts. Moreover, the low scores reflect the challenging nature of the problem of setting ambiguity. Every participant assesses ambiguity differently, and it may be debatable to have objective decisions across groups. A low score represents the cases where not all participants agree on a label, which is a realistic situation in many real world settings.

VI. Experiments

We organize our evaluation around three main questions. First, can our system automatically generate ambiguity metadata of good quality w.r.t. the manual annotation of the users? Second, are a-queries an effective solution for the automatic generation of large corpora? Third, can the generated sentences with row and attribute ambiguity significantly increase the quality of target applications?

Settings. For our experiment, we use as a pre-trained model the Text-to-Text Transfer Transformer (or T5) [18] pre-trained on the "Colossal Clean Crawled Corpus" [18]. T5 has state-of-the-art results on multiple NLP tasks including text generation which is the task we use to model the ambiguity metadata prediction. Experiments have been executed on a MacBook Pro with an Intel i9 CPU@2.9Ghz and 32 GB of memory. We used PostgresSQL 14.1 as the underlying DBMS.

A. Quality of Ambiguity Metadata

We now analyze in detail the solution proposed to identify attributes with ambiguities and their corresponding labels

TABLE III. RESULTS IN % FOR ALL METHODS ON THE TEST DATASET. QUALITY OF PREDICTING AMBIGUITY AND A LABEL FOR A GIVEN PAIR OF ATTRIBUTES. OUR METHODS REPORTED AT THE BOTTOM.

	Ambiguity			**Labeling**		
	P	R	F1	P	R	F1
uLABEL	**89.7**	10.3	18.5	84.2	6.4	6.35
sLABEL	85.6	86.1	85.9	74.1	41.5	53.3
SCHEMA	88.8	84.9	**86.8**	**87.8**	77.4	**82.3**
DATA	80.4	**91.3**	85.5	79.2	**84.5**	81.8

(Section III). In the following, we refer to our methods described in Section III-C as SCHEMA and DATA.

Datasets. For the training, we take a sample of 500k tables from the Web Tables corpus [17]. We use relational tables with a header as first row and horizontal orientation. For the test, we use the annotated corpus described in Section V. For queries over ConceptNet and Wikipedia, we use their online APIs.

Metrics. We do not report accuracy, defined as the fraction of correct predictions. As the number of pairs of attributes defined as ambiguous is much smaller than the number of attribute combinations, the label distribution is skewed (output is dominated by true negative). We therefore report precision (P), recall (R), and their combination in the f-measure (F1) for the prediction of the models, where a prediction for a label is a true positive if such label is in the ground truth.

Baselines. The annotator functions introduced in Section III-B perform very poorly when naively applied to the test dataset. We therefore introduce two baseline methods. The first, is an unsupervised labeling heuristic function, uLABEL, that uses both ConceptNet and Wikipedia to find possible common words for the attributes with intersection. If the result is still empty, then it returns the output of the *LCS* function. The second is a supervised labeling solution (sLABEL), that also makes use of the pre-trained LM. This fine tuning task takes a single attribute as input and produces a list of possible labels. It starts with the examples from the same annotators, i.e., synonyms of the attribute, ConceptNet labels, Wikipedia page titles, and the least common sub-string between the attribute and every other attribute in the table. The resulting list of possible labels is then used as training data. For testing, each attribute is submitted to the model and the pairs of attribute with non-empty intersection of their outputs are added to the results.

For the evaluation, we start by comparing the different methods. We then show how different parameters have an impact on our solution. All results are reported for the binary task of stating if two attributes are ambiguous (**Ambiguity**) and the task of predicting the correct label (**Labeling**).

Results w.r.t. the baselines. Table III shows the results for the four methods on the test corpus. We observe that the unsupervised baselines obtains good precision in both tasks, but very low recall. The other methods perform all well in terms of detecting ambiguous pairs (Ambiguity), with sLABEL close to our methods in terms of f-measure thanks to very high precision and recall. However, in the task of predicting the label, both our models clearly outperform both baselines with an f-measure of 82%, while sLABEL achieves only 53%. Interestingly, both models not using data (sLABEL and SCHEMA) achieve high precision, while the model that uses schema and data (DATA) achieves much higher recall. This is because the comparison of data values may lead to ambiguities that are not captured by looking at attribute label only. However, this may be misleading in some cases, such as those with numerical values with similar domains but different value distributions. For example, for attributes FG_PCT and FG3M, the human annotators agree on 'FG' as ambiguous label, but DATA returns *none* as they have different value distributions.

For DATA, we found experimentally that the best quality results are achieved with the maximum number of rows to consider in the T_{sd} equals to five and that the row representation outperforms the column representation.

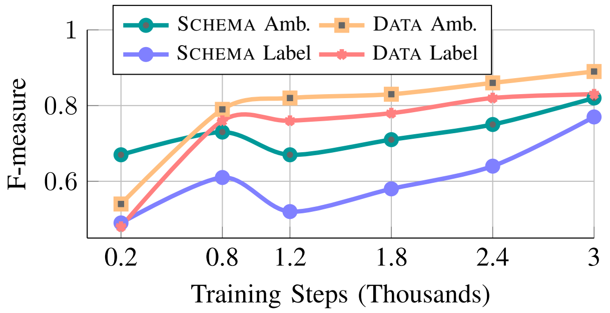

Fig. 6. Impact of the number of training step on prediction quality.

Results w.r.t. the number of training steps. Figure 6 shows that both methods improve the quality of the results with an increasing number of training steps. Results nearly converge after 3k steps. Training SCHEMA and DATA models requires 1.5 hours.

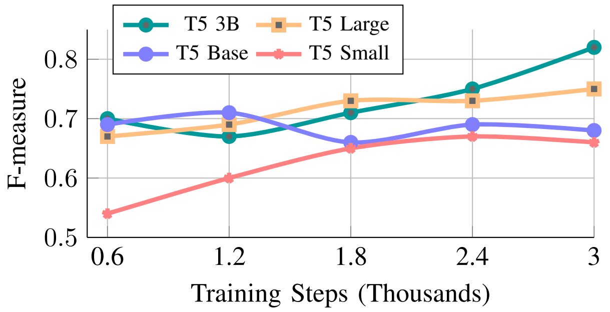

Fig. 7. Impact of the number of T5 parameters on prediction quality.

Results w.r.t. the number of T5 parameters (model size). Figure 7 reports the impact of the number of training parameter on the quality of the results. Increasing the size of the model parameters (sizes: Small < Base < Large < 3B) increases the quality of the predictions in the final model. LMs with a small number of parameters after 2000 training steps start to converge to low-quality results. The number of parameters also influences the inference time. The biggest model (3B) takes on average two seconds per prediction on our machine. We leave the exploration of recent methods for reducing time complexity in transformers to future work [38]

Results w.r.t. number of tables in the training table corpus. We experimented with a varying number of tables sampled from the Web Tables corpus. We executed annotators on sample of increasing size, from 5k to 500k tables. Experiments show that the quality marginally increases with larger datasets, but, even with a small sample, the annotators gather enough training data to train the model, both for SCHEMA and for DATA.

TABLE IV. NUMBER OF GENERATED EXAMPLES, BY TEXT GENERATION AND BY TEMPLATES, WITH ALL AMBIGUITY TYPES.

Dataset			**Text Gen.**		**Templates**		
	# rows	# atts	# exs	time (h)	# qrys	# exs	time (sec)
Iris [34]	150	6	1.5k	1.3	18	46.8k	0.2
Abalone [34]	1k	10	42k	11.6	54	1.6M	22.3
Adult [34]	1k	15	36k	10.1	13	1.7M	30.3
Basket [35]	538	27	54k	45	188	1.8M	31.7
$\text{Basket}_{Attribute}$	538	27	29k	24.3	65	1.7M	31.5
Basket_{Row}	538	27	8.1k	6.8	59	27.2k	0.3
Basket_{Full}	538	27	17k	14.2	63	7.3k	0.2
Mushroom [34]	100	24	9.4k	7.8	46	115k	0.5
Mushroom [34]	250	24	23k	19.5	46	726k	5.4
Mushroom [34]	500	24	45k	48.3	64	3.0M	30.2
Mushroom [34]	1k	24	83k	69.2	64	12.6M	175.3

B. Example Generation

We start by showing how both text generation and templates create a large number of ambiguous examples with our solution (Section IV). We report in Table IV the number of examples and execution time for the two approaches (plus the number of a-queries for templates). The table details the results for five tables in our corpus with different settings. Attribute ambiguities are obtained with SCHEMA in its default configuration (500k web tables used for training). PYTHIA in this experiment is set to produce only contradicting examples.

We observe that a large number of sentences are generated even for very small datasets, such as 1.5k and 47k for only 150 rows in Iris with text generation and templates, respectively. Similarly, we report 36k and1.7M for 1k rows sampled from the Adult dataset. We show with the Mushroom dataset that, by increasing the number of tuples in the sample from 100 to 1k, the number of output sentences increases at least by one order of magnitude. For the Basket dataset, we also report a breakdown of the impact of the different structure ambiguity type (Attribute, Row, Full) by running the same experiment over a fixed sample. The attribute ambiguity leads to more sentences than the other ambiguities in orders of magnitude. This supports our effort in predicting ambiguous attributes and their label to support this ambiguity type. We also remark that all output examples derived from templates are generated from the predefined three templates ($Q1, Q2, Q3$) with no human intervention or annotation.

The different execution times between the text generation and templates strategies are due to the use of the text generator in the first case. Templates rely only on standard SQL queries to produce both evidence and text, while the text generation approach requires the execution of a large language model for inference, which takes significant time and leads to a smaller number of generated examples. Despite this drawback, in the following we report experiments with examples generated with our default approach, the text generation, as it is fully automatic. We show next that the target applications can start benefiting even with a small number of ambiguity examples.

C. Impact on Target Applications

In this section, we show the benefits of automatically generating training corpora with ambiguous annotated sentences in two fact-checking systems, Feverous and CoronaCheck, and in a Natural Language to SQL tool. We use the ambiguity metadata and the examples automatically generated by PYTHIA without any human intervention.

Feverous. Computational fact checking aims at verifying if a given claim holds w.r.t. some reference information; we focus on the check done over relational tables. Feverous [4] is a dataset for this task containing 87k textual claims. Every claim is annotated with three possible labels by human annotators. A claim is labelled as *Support* or *Refute*, according to the evidence in the data or *NEI* for not enough evidence from the data. This experiment is designed to show that, by using PYTHIA's examples, we can improve the results for each class without changing the proposed baseline models[5].

TABLE V. RESULTS ON F_{test} CORPUS WITH FEVEROUS SYSTEM.

	CLASS	**P**	**R**	**F1**
Feverous (baseline)	NEI	0.50	0.34	0.40
	Supports	0.68	**0.78**	0.72
	Refutes	0.72	0.78	0.75
Feverous on F_t + P_t	NEI	**0.62**	**0.59**	**0.60**
	Supports	**0.77**	0.73	**0.75**
	Refutes	**0.80**	**0.85**	**0.82**

Since there is no golden standard for the test set, we used the provided evaluation dataset. We sampled the evaluation dataset in two disjoint datasets. A training dataset F_t consists of 1.1k examples (223 NEIs, 388 Supports, 489 Refutes), and a test dataset F_{test} consists of 276 examples (57 NEIs, 98 Supports, 121 Refutes). To have a fair comparison, we fine-tune the Feverous baseline model with F_t for 5 epochs and measure P, R and F1 for each class.

Results are reported in Table V. The NEI class is the one with lowest accuracy in the original model. This is due to the challenges of the class itself, since the evidence cells in many cases do not contain any informative value or contain partial data without enough evidence. Our goal is to show that PYTHIA examples can help in increasing the quality of the NEI classification. The intuition is that ambiguous examples are indeed NEI examples, and the system should improve in recognizing them with PYTHIA's output. Using the datasets from Table IV, we generate a set of new ambiguous examples (sentences, evidence cells) P_t. We use the original F_t and the new P_t to fine-tune the baseline with 5 epochs. We tested different sizes of P_t but the best results are obtained with 1240 examples (around 50% of the training data is ambiguous), as with more examples the model starts to overfit.

Table V shows that the PYTHIA's examples increase the F1 for all classes, and up to 60% (from 40%) for the challenging NEI class. In a previous experiment [12], we also show that the Feverous model can be extended to learn how to classify claims with data ambiguities. In that experiment, we fine-tune the baseline with PYTHIA generated data to let the model learn a new label. We remark the different test sets. In this experiment, we show that we can improve the results of the original Feverous data, while in the previous one we use only PYTHIA generated data to "teach" a new label.

CoronaCheck. As a second fact checking application, we show

[5]https://github.com/Raldir/FEVEROUS/

the verification of statistical claims related to COVID-19 [6]. We use PYTHIA's examples to improve an existing system (https://coronacheck.eurecom.fr). The original system is not trained to handle attribute ambiguity. Also, row ambiguity causes the original system to hallucinate in some cases, with lower classification accuracy for ambiguous claims. One of the tables used for verifying the statistical claims includes the following attributes: *country*, *date*, *total_confirmed*, *new_confirmed*, *total_recovered*, *active_cases total_fatality_rate*, *total_mortality_rate*. From the analysis of the log of the claims submitted by users, an example of common attribute ambiguity is between attributes *total_fatality_rate* and *total_mortality_rate* for sentences containing "death rate". Another example of attribute ambiguity is between *total_confirmed* and *new_confirmed* when sentences only mention "cases". Examples of row ambiguity consist of claims that refer to two records with same location but different timestamps, such as "In France, 10k confirmed cases have been reported" (today, yesterday or last week?), or when location is not present in the claim, such as "35000 new covid cases today" (US, China, or World?).

TABLE VI. IMPACT OF P_t DATA ON CORONACHECK.

Ambiguity	Users' Claims	Accuracy original	Accuracy original+PYTHIA
Row	40	32/40	**34**/40
Attribute	8	0/8	**7**/8
Full	40	0/40	**27**/40
None	12	7/12	**7**/12
Total	100	39/100	**75**/100

PYTHIA enables us to improve CoronaCheck by automatically generating ambiguity aware training data. We generate, with the text generator module and with templates, a new corpus of examples that include all ambiguity structure types. The new corpus is then merged with the not ambiguous examples in a 50/50 ratio and used to train new classifiers that allow the extension of the original system to recognize ambiguity. For this experiment, we collect a sample of 100 claims from the log of CoronaCheck as they have been submitted by users. As shown in Table VI, 88% include at least a row or an attribute ambiguity. Over all claims: 40% have exclusively row ambiguity, 8% have exclusively attribute ambiguity, and 40% have both attribute and row ambiguities, i.e., full ambiguity. The original CoronaCheck fails to classify claims with ambiguity. However, training the system with the output of PYTHIA leads to significant enhancements in accurately handling all cases without a drop in quality in the claims without ambiguity.

We perform an error analysis by checking the examples where both systems fail for the ambiguity types in Table VI. For row ambiguity, it is either due to the unsupported nature of the claim ("An exponential increase in coronavirus infections has been recorded in France", more complex aggregation required) or to the misclassification in the system classifiers, such as the attribute one ("in Northern Ireland, taking cumulative total to 7,294.", where it predicts recovered instead of confirmed). This is also the case for the attribute ambiguity, for "Were there 6000 cases in Switzerland today?" the system predicts to consider exclusively total_confirmed instead of new_confirmed and active_cases, and for the full ambiguity, e.g., for "In Lebanon with a population of <5M, 8300 #COVID19 cases.", it ignores verifying active_cases. For the claims that are not ambiguous, there are also errors due to the presence of complex operations, such as aggregation. Such cases represent 5% of the annotated claims. As examples: "The maximum number of daily new confirmed cases in Italy during July 2021 was 420,123." and "January 17, 2022: After the issue of the pass vaccinal law, a record of new vaccination was observed in France since the beginning of 2022.".

TABLE VII. TEXT-TO-SQL RESULTS ON TEST CORPUS

	Train Size	P	R	F1	ACC.	BLEU
Baseline	WikiSQL	-	-	-	0.54	0.49
FT$_{Pythia}$	+200	0.62	0.92	0.74	0.71	20.47
FT$_{Pythia}$	+481	0.75	0.78	0.76	0.78	28.19
FT$_{Pythia}$	+2207	0.82	0.88	0.85	0.86	**38.70**
FT$_{Pythia}$	+6227	**0.96**	0.73	0.83	0.87	26.50
FT$_{Pythia}$	+10219	0.88	**0.94**	**0.91**	**0.92**	32.13

To see the benefit of using PYTHIA compared to the original system, consider the example with "June 2021: France has 111,244 Covid-19 deaths.". The claim is True for total_deaths and False for new_deaths. However, the original system returns a false decision by checking against *new_deaths* only. With PYTHIA's training data, the predictions of the checking system are True for *total_deaths* and False otherwise.

Text-to-SQL. WikiSQL [39] is a crowd-sourced dataset containing natural language questions and their relative SQL queries over Wikipedia tables. We use a T5 base pre-trained model on WiKiSQL[6] on PYTHIA's generated dataset.

The generated dataset is split between queries with and without ambiguities and we used both text generation and template based approach (over our tables in Section V) to scale up with the training size.

For the training set, we use tables Adults, Soccer, Laptop, and Hearth Diseases (Kaggle). For the test set, we use tables Abalone, Iris, Wine Quality (Kaggle) and the two versions of Basket. Since a single query could generate a different number of examples, and to keep the generated datasets with high variance, we use only a single question for each query.

We test the ability of the model to generate a *query*, if there is no ambiguity in the input NL question, or to return *none* otherwise. To evaluate the results, we measure the accuracy (ACC.) and the BLEU scores of the generated queries w.r.t. the expected query in the ground truth. BLEU measures the number of words in the generated SQL query that appears in the labelled SQL query. Higher values of the metric indicate better quality of the generated SQL query.

We compare two models: the pre-trained T5 as baseline and a fine-tuned version of the same pre-trained model on PYTHIA generated examples (FT$_{Pythia}$). For the fine-tuned models, we also report the precision (P), recall (R), and f-measure (F1) in detecting the ambiguity. To evaluate the impact of the number of new examples, we report the performances when increasing the size of random samples from the generated training set.

Results are reported in Table VII. The best results are reported in bold. The base model fails on all the ambiguous examples, with a very low BLEU score, and the best results

[6]https://huggingface.co/mrm8488/t5-base-finetuned-wikiSQL

are always obtained by the fine-tuned model. Fine-tuning the model with more training data increases the accuracy and the F1 metric. We obtain the best results fine-tuning the T5 with 10.2k examples. Also, the BLEU score benefits from more training size with a peak at 2.2k examples.

TABLE VIII. QUALITY RESULTS OF THE END-TO-END EXPERIMENT

Dataset	Ambiguity			Attr. Ambiguity		
	P	R	F1	P	R	F1
Abalone	1.000	0.807	0.893	1.000	0.807	0.893
Adults	0.923	0.889	0,906	0.885	0.885	0.885
Basket Acronyms	0.750	0.813	0.780	0.731	0.809	0.768
Basket	1.000	0.619	0.765	1.000	0.619	0.765
Heart Diseases	0.875	0.656	0.750	0.833	0.645	0.727
Iris	1.000	0.963	0.981	0.981	0.962	0.971
Superstore	0.950	0.679	0.792	0.900	0.667	0.766
Wine Quality	0.950	0.792	0.864	0.950	0.792	0.864
Laptop	0.923	0.667	0.774	0.846	0.647	0.733
Mushroom	1.000	0.905	0.950	1.000	0.905	0.950
Soccer	0.762	0.800	0.780	0.714	0.789	0.750
AVG	0.921	0.781	0.840	0.895	0.775	0.825

D. End-To-End User Evaluation

We conduct a second user study to evaluate the text generated by PYTHIA over 11 datasets. We generate at least four ambiguous sentences (half with text generation and half with templates) and two not ambiguous ones for each table. We ask eleven participants to manually annotate the generated text in two ways. First, *ambiguity detection*: state if the text is ambiguous w.r.t. the associated schema and a sample of data. Second, *attribute ambiguity detection*: if the text is ambiguous, mark the ambiguous attributes. We instruct the participants with data ambiguity definitions and examples. We give each participant three datasets to annotate, so that every dataset has three annotations. Then, for every annotator and dataset, we measure P, R and F1 for both ambiguity and attribute ambiguity detection. For ambiguity, we count a match if the annotation agrees with the binary label for the text in the ground truth. For the attribute ambiguity, we count a match if at least one of the annotated attributes is in the ground truth of the text. Results per dataset (averaged over three participants) are shown in Table VIII. We observe an average F1 (over all datasets) of 84% and 82.5% for ambiguity and attribute ambiguity detection, respectively. Such results show that (1) humans recognize text with and without data ambiguity, (2) they also recognize the right attributes when there is ambiguity. As expected, the second action has lower F1 as it is an harder task.

VII. RELATED WORK

A popular solution to extend training text data is augmentation [40], [41]. However, these methods create variations of existing examples, e.g., replace a name in the text with a synonym, and do not consider the problem of creating examples with data ambiguity.

Controllable NL generation approaches steer neural models to generate text with desired attributes by training the decoding algorithm [42], training a conditioned language model [43], or fine-tuning with reinforcement learning [44]. However, such approaches assume the ability to quantitatively measure the desired property (such as positive sentiment) or metric (such as BLEU), which is not the case for ambiguity.

Text generation to get training data for fact checking is also related [45], but does not cover ambiguities in claims. In the context of fact checking, there are studies on how to modify a given query, representing a textual claim [7]. Given a claim c represented as a parameter template query q, they find the query parameters that make the claim plausible. However, the query is given in this setting. Beyond fact checking and text-to-SQL, our proposal is useful for query answering applications that need ambiguous textual questions for training [9].

There has been work on verbalizing tables to produces sentences that can be logically entailed by the associated table [46]. However, their statements are based on the TabFact dataset that has been annotated excluding any type of ambiguity [31]. Our approach is also distinct from those that verbalize knowledge graphs through the use of semantic web technologies, such as RDF [47], [48], OWL [49], [47], [50] and SPARQL [51]. Those are rich in metadata, with clear semantic relationships, and existing efforts on resolving ambiguities focus only on lexicology [47], [51].

As we are interested in ambiguity w.r.t. the content of the relation, we focus on texts that contain facts from the cell values only, without introducing further uncertainty in the values themselves, i.e., we are not focusing on challenges related to matching tokens or entities [52], [53]. Finally, our work is related to skyline queries and groups [54], [55], [56]. However, such line of work assumes the attributes to be available and in our case we must identify the ambiguous attributes.

VIII. CONCLUSION

We have proposed a solution that automatically constructs high-quality datasets for machine learning. Experiments show that given only a relational table as input, PYTHIA identifies the ambiguity metadata with very high precision and recall. With such metadata, the system automatically generates rich examples, composed of text together with its relevant table cells, that effectively train target applications to handle text with data ambiguity.

While handling ambiguity is an important first step, there are several other classes of under-represented examples in NLP corpora that could benefit from their automatic generation from relational data. Such examples include mathematical operations, text that span multiple tables, and non-factual text, such as "What is the player with highest shooting?". For instance, consider again two tables *Covid (country, date, vaccinated, positive)* and *Regions (region, country)* with the sentence "The total number of vaccinated in EU is higher than in Africa". While manually crafting a template to generate the queries for this sentence is always possible (it would include a comparison of two sums over two groups that are derived from the join of the tables), it would be very valuable to have a method that synthesizes the template given only the tables and the example. Other directions include an (1) interactive version of the system, (2) new algorithms to exploit correlation across ambiguous attributes, (3) the design of new text augmentation techniques that make use of ambiguity metadata, and (4) profiling methods that use both attributes labels and value overlaps.

References

[1] [n.d.], "Notes on ambiguity," https://cs.nyu.edu/∼davise/ai/ambiguity.html, 2021.

[2] BleacherReport, "Stephen Curry is 'the best shooter who ever lived' per Warriors HC Steve Kerr," https://bleacherreport.com/articles/10009994, 2021.

[3] G. Katsogiannis-Meimarakis and G. Koutrika, "A deep dive into deep learning approaches for text-to-sql systems," in *SIGMOD*. ACM, 2021, pp. 2846–2851.

[4] R. Aly, Z. Guo, M. S. Schlichtkrull, J. Thorne, A. Vlachos, C. Christodoulopoulos, O. Cocarascu, and A. Mittal, "FEVEROUS: Fact extraction and VERification over unstructured and structured information," in *Thirty-fifth Conference on Neural Information Processing Systems (NIPS - Datasets and Benchmarks)*, 2021.

[5] N. Hassan, G. Zhang, F. Arslan, J. Caraballo, D. Jimenez, S. Gawsane, S. Hasan, M. Joseph, A. Kulkarni, A. K. Nayak, V. Sable, C. Li, and M. Tremayne, "Claimbuster: The first-ever end-to-end fact-checking system," *Proc. VLDB Endow.*, vol. 10, no. 12, pp. 1945–1948, 2017.

[6] G. Karagiannis, M. Saeed, P. Papotti, and I. Trummer, "Scrutinizer: A mixed-initiative approach to large-scale, data-driven claim verification," *Proc. VLDB Endow.*, vol. 13, no. 11, pp. 2508–2521, 2020.

[7] Y. Wu, P. K. Agarwal, C. Li, J. Yang, and C. Yu, "Computational fact checking through query perturbations," *ACM Trans. Database Syst.*, vol. 42, no. 1, pp. 4:1–4:41, 2017.

[8] W. Chen, M. Chang, E. Schlinger, W. Y. Wang, and W. W. Cohen, "Open question answering over tables and text," in *ICLR*. OpenReview.net, 2021.

[9] J. Thorne, M. Yazdani, M. Saeidi, F. Silvestri, S. Riedel, and A. Y. Levy, "From natural language processing to neural databases," *Proc. VLDB Endow.*, vol. 14, no. 6, pp. 1033–1039, 2021.

[10] A. P. Parikh, X. Wang, S. Gehrmann, M. Faruqui, B. Dhingra, D. Yang, and D. Das, "Totto: A controlled table-to-text generation dataset," in *EMNLP*. ACL, 2020, pp. 1173–1186.

[11] L. Massarelli, F. Petroni, A. Piktus, M. Ott, T. Rocktäschel, V. Plachouras, F. Silvestri, and S. Riedel, "How decoding strategies affect the verifiability of generated text," in *EMNLP*, vol. EMNLP 2020. ACL, 2020, pp. 223–235.

[12] E. Veltri, D. Santoro, G. Badaro, M. Saeed, and P. Papotti, "Pythia: Unsupervised generation of ambiguous textual claims from relational data," in *Proceedings of the 2022 International Conference on Management of Data*. Association for Computing Machinery, 2022, p. 2409–2412.

[13] A. Ng, L. Aroyo, C. Coleman, G. Diamos, V. J. Reddi, J. Vanschoren, C.-J. Wu, and S. Zhou, Eds., *Data-Centric AI Workshop*. NeurIPS, 2021.

[14] "PYTHIA code and datasets," https://github.com/enzoveltri/pythia, 2022.

[15] I. Trummer, "Data vocalization with CiceroDB," in *CIDR*. www.cidrdb.org, 2019.

[16] Z. Abedjan, L. Golab, F. Naumann, and T. Papenbrock, *Data Profiling*, ser. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2018.

[17] O. Lehmberg, D. Ritze, R. Meusel, and C. Bizer, "A large public corpus of web tables containing time and context metadata," in *WWW*. ACM, 2016, pp. 75–76.

[18] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res.*, vol. 21, pp. 140:1–140:67, 2020.

[19] A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré, "Snorkel: Rapid training data creation with weak supervision," in *VLDB*, vol. 11, no. 3, 2017, p. 269.

[20] R. Speer, J. Chin, and C. Havasi, "Conceptnet 5.5: An open multilingual graph of general knowledge," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[21] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2019, pp. 4171–4186. [Online]. Available: https://doi.org/10.18653/v1/n19-1423

[22] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[23] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *Advances in neural information processing systems*, vol. 32, 2019.

[24] Y. Goldberg and O. Levy, "word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method," *arXiv preprint arXiv:1402.3722*, 2014.

[25] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[26] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "Fasttext. zip: Compressing text classification models," *arXiv preprint arXiv:1612.03651*, 2016.

[27] G. Badaro, M. Saeed, and P. Papotti, "Transformers for tabular data representation: A survey of models and applications," in *EURECOM*, 2021, technical Report, October 2021: https://www.eurecom.fr/publication/6721.

[28] P. Yin, G. Neubig, W.-t. Yih, and S. Riedel, "Tabert: Pretraining for joint understanding of textual and tabular data," *arXiv preprint arXiv:2005.08314*, 2020.

[29] J. Herzig, T. Müller, S. Krichene, and J. Eisenschlos, "Open domain question answering over tables via dense retrieval," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2021, pp. 512–519. [Online]. Available: https://doi.org/10.18653/v1/2021.naacl-main.43

[30] F. Pan, M. Canim, M. Glass, A. Gliozzo, and P. Fox, "Cltr: An end-to-end, transformer-based system for cell level tableretrieval and table question answering," *arXiv preprint arXiv:2106.04441*, 2021.

[31] W. Chen, H. Wang, J. Chen, Y. Zhang, H. Wang, S. Li, X. Zhou, and W. Y. Wang, "Tabfact: A large-scale dataset for table-based fact verification," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=rkeJRhNYDH

[32] M. Kale and A. Rastogi, "Text-to-text pre-training for data-to-text tasks," in *Proceedings of the 13th International Conference on Natural Language Generation, INLG*. Association for Computational Linguistics, 2020, pp. 97–102.

[33] E. Veltri, G. Badaro, M. Saeed, and P. Papotti, "Pythia: Generating Ambiguous Sentences From Relational Data," https://www.eurecom.fr/∼papotti/pythiaTr.pdf, 2022.

[34] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[35] S. Wiseman, S. Shieber, and A. Rush, "Challenges in data-to-document generation," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sep. 2017, pp. 2253–2263. [Online]. Available: https://aclanthology.org/D17-1239

[36] K. Krippendorff, "Computing krippendorff's alpha-reliability," in *University of Pennsylvania*, 2011.

[37] I. Mozetič, M. Grčar, and J. Smailović, "Multilingual twitter sentiment classification: The role of human annotators," *PloS one*, vol. 11, 2016.

[38] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," *ACM Comput. Surv.*, apr 2022. [Online]. Available: https://doi.org/10.1145/3530811

[39] V. Zhong, C. Xiong, and R. Socher, "Seq2sql: Generating structured queries from natural language using reinforcement learning," *CoRR*, vol. abs/1709.00103, 2017.

[40] S. Y. Feng, V. Gangal, D. Kang, T. Mitamura, and E. Hovy, "GenAug: Data augmentation for finetuning text generators," in *DeeLIO*, 2020, pp. 29–42.

[41] R. Liu, G. Xu, C. Jia, W. Ma, L. Wang, and S. Vosoughi, "Data boost: Text data augmentation through reinforcement learning guided conditional generation," in *EMNLP*, B. Webber, T. Cohn, Y. He, and Y. Liu, Eds. Association for Computational Linguistics, 2020, pp. 9031–9041.

[42] J. Gu, K. Cho, and V. O. Li, “Trainable greedy decoding for neural machine translation,” in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sep. 2017, pp. 1968–1978. [Online]. Available: https://aclanthology.org/D17-1210

[43] N. S. Keskar, B. McCann, L. Varshney, C. Xiong, and R. Socher, “CTRL - A Conditional Transformer Language Model for Controllable Generation,” *arXiv preprint arXiv:1909.05858*, 2019.

[44] M. Ranzato, S. Chopra, M. Auli, and W. Zaremba, “Sequence level training with recurrent neural networks,” in *ICLR*, 2016.

[45] L. Pan, W. Chen, W. Xiong, M. Kan, and W. Y. Wang, “Zero-shot fact verification by claim generation,” in *ACL/IJCNLP 2021*. Association for Computational Linguistics, 2021, pp. 476–483. [Online]. Available: https://doi.org/10.18653/v1/2021.acl-short.61

[46] W. Chen, J. Chen, Y. Su, Z. Chen, and W. Y. Wang, “Logical natural language generation from open-domain tables,” in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, Jul. 2020, pp. 7929–7942. [Online]. Available: https://aclanthology.org/2020.acl-main.708

[47] D. M. Axel-Cyrille Ngonga Ngomo and L. Bühman, “A Holistic Natural Language Generation Framework for the Semantic Web,” in *International Conference Recent Advances in Natural Language Processing*. ACL, 2019.

[48] L. Perez-Beltrachini, R. Sayed, and C. Gardent, “Building RDF content for data-to-text generation,” in *COLING*, Dec. 2016.

[49] I. Androutsopoulos, G. Lampouras, and D. Galanis, “Generating natural language descriptions from owl ontologies: The natural owl system,” *J. Artif. Int. Res.*, vol. 48, no. 1, p. 671–715, Oct. 2013.

[50] N. Bouayad-Agha, G. Casamayor, and L. Wanner, “Natural language generation in the context of the semantic web,” *Semantic Web*, vol. 5, pp. 493–513, 2014.

[51] A.-C. Ngonga Ngomo, L. Bühmann, C. Unger, J. Lehmann, and D. Gerber, “Sorry, i don’t speak sparql: Translating sparql queries into natural language,” in *WWW*, ser. WWW ’13. Association for Computing Machinery, 2013, p. 977–988.

[52] P. Suganthan, A. Ardalan, A. Doan, and A. Akella, “Smurf: Self-service string matching using random forests,” *PVLDB*, vol. 12, no. 3, pp. 278–291, 2018.

[53] M. Ebraheem, S. Thirumuruganathan, S. Joty, M. Ouzzani, and N. Tang, “Distributed representations of tuples for entity resolution,” *PVLDB*, vol. 11, no. 11, pp. 1454–1467, 2018.

[54] S. Borzsony, D. Kossmann, and K. Stocker, “The skyline operator,” in *Proceedings 17th International Conference on Data Engineering*, 2001, pp. 421–430.

[55] N. Zhang, C. Li, N. Hassan, S. Rajasekaran, and G. Das, “On skyline groups,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 4, pp. 942–956, 2014.

[56] H. Zhu, X. Li, Q. Liu, and Z. Xu, “Top-k dominating queries on skyline groups,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 7, pp. 1431–1444, 2020.