Generation of Training Examples for Tabular Natural Language Inference

Jean-Flavien Bussotti
EURECOM, France

Enzo Veltri
University of Basilicata, Italy

Donatello Santoro
University of Basilicata, Italy

Paolo Papotti
EURECOM, France

ABSTRACT

Tabular data is becoming increasingly important in Natural Language Processing (NLP) tasks, such as Tabular Natural Language Inference (TNLI). Given a table and a hypothesis expressed in NL text, the goal is to assess if the former structured data supports or refutes the latter. In this work, we focus on the role played by the annotated data in training the inference model. We introduce a system, Tenet, for the automatic augmentation and generation of training examples for TNLI. Given the tables, existing approaches are either based on human annotators, and thus expensive, or on methods that produce simple examples that lack data variety and complex reasoning. Instead, our approach is built around the intuition that SQL queries are the right tool to achieve variety in the generated examples, both in terms of data variety and reasoning complexity. The first is achieved by evidence-queries that identify cell values over tables according to different data patterns. Once the data for the example is identified, semantic-queries describe the different ways such data can be identified with standard SQL clauses. These rich descriptions are then verbalized as text to create the annotated examples for the TNLI task. The same approach is also extended to create counter-factual examples, i.e., examples where the hypothesis is false, with a method based on injecting errors in the original (clean) table. For all steps, we introduce generic generation algorithms that take as input only the tables. For our experimental study, we use three datasets from the TNLI literature and two crafted by us on more complex tables. Tenet generates human-like examples, which lead to the effective training of several inference models with results comparable to those obtained by training the same models with manually-written examples.

ACM Reference Format:
Jean-Flavien Bussotti, Enzo Veltri, Donatello Santoro, and Paolo Papotti. 2023. Generation of Training Examples for Tabular Natural Language Inference. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In Natural Language Processing (NLP), a large class on natural language inference (NLI) problems aim at classifying a given hypothesis, such as a textual statement, as true/false/unknown given some evidence. While this is a well-studied problem for the setting with text as evidence [41], recently it has emerged a new class of applications that focus on inference with structured data as evidence, i.e., *tabular natural language inference* (TNLI). Example applications are table understanding [19, 20] and computational fact checking, where systems label text claims according to input structured data [22, 35, 54][1].

The best solutions for TNLI are supervised. Manually defined datasets for TNLI have been proposed, such as Feverous [2], TabFact [11], and Infotabs [19]. However, these datasets have three main issues. (i) They cover only some generic topics with tables from Wikipedia. For example, if there is a need for fact-checking claims for emerging domains such as Covid-19, a new annotated corpus must be crafted by manually writing examples using the tabular reports published by governments. (ii) They are not comparable in scale and variety to those available for textual NLI [41]. In terms of reasoning requirements, about 80% of the examples in Totto [39] have sentences describing the data with text that does not contain mathematical expressions, such as max, min and count, or comparison across values. (iii) They contain bias and errors that may lead to incorrect learning in the target models [18].

The problem of the lack of labeled examples has been treated in the literature for NLI, but it has not been tackled yet for TNLI. If some examples are given, in a *warm start* setting, existing NLI augmentation methods can be used in the TNLI setting: the text part of the example can be rewritten with augmentation w.r.t. the (fixed) data [6]. While these methods increase the number of examples, they do not generate a new corpus that raises the variety and complexity of the examples w.r.t. the structured data, ultimately with minor impact on the accuracy of the TNLI tasks. Moreover, in a *cold start* setting, where training data is not available, there is no proposal yet on how to create annotated examples for TNLI starting only from the tables.

In this work, we argue that user provided tables can be exploited to generate ad-hoc training data for the application at hand. Our system, Tenet[2] (TExtual traiNing Examples from daTa) generates large annotated corpora of training examples that are complex and rich in terms of data patterns, linguistic diversity, and reasoning complexity. Figure 1 shows an overview of our proposed method. The system generates training data for the target TNLI application

[1] E.g., https://coronacheck.eurecom.fr
[2] Code and datasets available at https://github.com/dbunibas/tenet

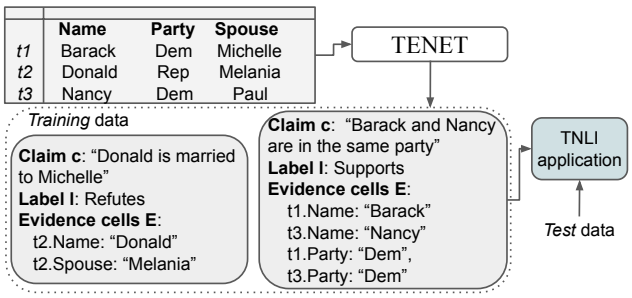

Figure 1: Given any table, Tenet generates new training examples for a target TNLI application. The first example has a hypothesis that is refuted according to the data evidence.

given only a table as input. Once the examples are generated, they are used to train the inference model that is validated on test data.

Tenet is built around three modules that cover the three main elements of a complete and annotated TNLI example.

Data Evidence. A key intuition in our approach is that tabular data already contains rich information for new examples. Content changes across datasets, and every relation has its own active domain. Moreover, data relationships across entities and their properties are arranged differently across datasets. To identify *data evidence* to create a variety of examples, we propose alternative approaches to select sets of cells from the given table, including a query generation algorithm for the semi-supervised case. A query returns a set of evidence, such as *Donald* and *Michelle* in the first example in Figure 1, each partially describing an example.

Textual Hypothesis. Once the data is identified, we obtain the textual statement (or *hypothesis*) for the annotated example. Given a set of cells, we generate queries that identify such data evidence over the input table. Every query characterizes the data with different conditions (e.g., selections with constants) or constructs (e.g., aggregate). From the query and the evidence, we create a text with a prompting method that exploits the human-like generation abilities of large pre-trained language models (PLMs), such as GPT-3 [47]. Our prompting leads to a variety of hypothesis that are factual, such as *Barack and Nancy are in the same party* in the second example in Figure 1, while maximizing the coverage of the provided evidence and minimizing hallucination.

Inference Label. Finally, we need the corresponding *label* for every example. While Supports examples are obtained naturally, as the hypothesis reflects the evidence from the table, for Refutes examples we introduce generic methods built around the idea of injecting errors in the data evidence. Once the data is modified, the process for text generation is applied to the "dirty" data to obtain hypothesis that are refuted w.r.t. the original "clean" data.

Our contributions may be summarized in the following points:

- We introduce an end-to-end system that generates TNLI example from tabular data (Section 2). The system is generic, as it does not make assumptions w.r.t. the content of the input tables. The architecture supports both unsupervised generations (cold start) when only tables are provided, and semi-supervised (warm start), where some manually written examples are available. While the former is more general, the latter generates high-quality examples even in settings where the number of tables available for training is limited.
- We introduce algorithms for the generation of the three main components of an annotated example: data evidence (Section 3), textual hypothesis (Section 4), and Refutes counter-examples (Section 5). In every component we enforce variety in terms of data patterns and reasoning challenges.
- We show results for five TNLI test datasets, comparing the results obtained by training with manually written examples vs those obtained with training data generated by Tenet (Section 6). Training examples generated with Tenet lead to reasoning models that outperform the accuracy of the same models trained with data from human annotators in a variety of settings. We also show that Tenet's examples can be used in test data for the validation of reasoning models.

We then conclude the paper with a discussion of related work (Section 7) and open research directions (Section 8).

2 OVERVIEW OF THE SOLUTION

Problem Formulation. Let r be a tuple in the instance I for a relational schema R and A_i an attribute in R. We refer with *cell value* to the value of tuple r in attribute A_i and with *table* to the instance I for simplicity[3]. A *textual hypothesis* is a sentence in natural language.

A Tabular Natural Language Inference (TNLI) application takes as input a pair (table c; textual hypothesis h) and outputs if h is supported or refuted by c. *Data evidence* is a non empty subset of cell values from c that varies from a small fraction in some settings [2] to the entire relation in others [11][4]. Solutions for the TNLI task rely on supervised models trained with annotated examples - our goal is to reduce the effort in creating such training data.

We consider solving the *example generation problem* for a TNLI application A where we are given the label space L for A, a corpus of tables C, and (optionally) a set of training examples T for A. Every example is composed by a quadruple (h, l, e, c) with textual hypothesis h, label $l \in L$, set of data evidence cells e contained in one relational table c in the corpus C. We assume access to a text-to-text pre-trained language model (PLM) M. We do not assume access to the TNLI application A at hand. In this work we focus on L with *Supports* and *Refutes* labels only, as those are the most popular in TNLI corpora, e.g., 97% of the examples [2].

In the *warm start* version of the problem, training examples for A are available and used by Tenet. In the *cold start* version of the problem, we drop the assumption on the availability of the examples T. In this case, we aim at creating new training examples D for A just by using the tables in C.

Process and Challenges. Tenet is designed around three main steps, as depicted in Figure 2. Given a relation table $c \in C$, it first gathers the evidence (set of cells) e to produce a Supports example. Second, to enable the generation of a Refutes example, it injects errors in table c to create its noisy version and derive data evidence e'. Third, a textual claim (hypothesis) h is generated for every data

[3]Some TNLI corpora contain both relational and entity tables, i.e., relational tables transposed with a single row. Tenet supports both, but we focus the presentation on relational ones for clarity.

[4]Our proposal is independent from the size of the data evidence and its retrieval.

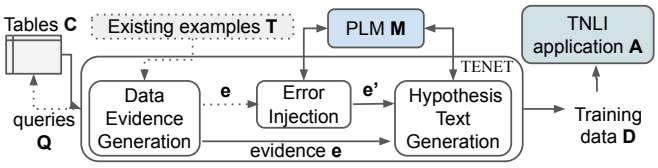

Figure 2: TENET overview. Existing examples are optional. Any text-to-text pre-trained language model (PLM) can be used, e.g., ChatGPT. Any target TNLI application can be supported, e.g., tabular fact-checking.

evidence e. The quadruple (data evidence e, textual claim h, label Supports/Refutes, table c) is a complete example for training data D for the target TNLI application A. However, the three steps come with their own challenges.

Table 1: People table. Cells in bold form data evidence e_1.

	Name	Age	City	Team
t_1	**Mike**	**47**	SF	DBMS
t_2	**Anne**	**22**	NY	AI
t_3	John	19	NY	DBMS
t_4	Paul	18	NY	UOL

Data Evidence. Training examples D must capture the variety of relationships in a table, such as those relating cell values in the same tuple or attribute. An hypothesis is defined over a group of cell values, such as the data evidence e_1 highlighted in bold in Table 1 for tuples t_1 and t_2, i.e., names of two people with different age values. Hypothesis "Mike is older than Anne" captures the relationship across these four cell values. A data evidence with two cell values, e.g., Name for tuple t_1 and Age from tuple t_2 can lead to an hypothesis, e.g., "There is a person called Mike and a person 22 years old", but such sentence does not capture relationships across tuples nor attributes. In general, for effective training, the data evidence covered by the examples should cover the variety of patterns that can be identified in a relation.

One approach for the data evidence generation is to pick different sets of cell values at random. While this simple approach is effective and enables an unsupervised solution, there are meaningful patterns, such as e_1, that may be covered rarely by accident. One approach to improve this task and obtain meaningful patterns with a smaller number of generated examples is to infer data patterns from human-provided examples T, when those are available. For an example in T, we identify a query q that returns the cell values in its data evidence as one result row. We then execute such query over the relation. The query leads to more sets of cells (one per result row) that enable the generation of examples following the same data pattern, for example involving t_3 and t_4.

Hypothesis. Given a table c and an evidence set $e \in c$, the latter can be described with a textual sentence. However, the *way* a set of cells is converted to a sentence has a huge impact on the variety and the reasoning complexity of the training data. Indeed, given a set of cells from a table, many alternatives exist for describing it in natural language. Consider again data evidence e_1 in the example. The values in bold can be correctly described with "Mike is older than Anne." or "There are two persons with age higher than 19.". The more alternative sentences for a given data evidence are created, the better the training set for the target model. Unfortunately, most efforts for automatic data-to-text are focused on *surface*, or *look-up*, sentences [39], such as "Mike is 47 years old and Anne 22.". While this kind of sentences are fundamental, we aim at maximizing the variety in the training data. For this goal, we generate various queries that return evidence e given c. Such queries represent different ways of semantically describing the data. We then propose prompting methods for PLMs to generate alternative sentences to describe the evidence set according to the semantics of the queries.

Label. By construction, the generated data evidence is coherent with the semantics expressed in the input table. An evidence set leads to an example with a Supports label w.r.t. the data in the table. However, applications also need examples with a Refutes label, i.e., textual claims not supported by the input table. We tackle this problem with an error injection approach, perturbing the input table to break the original relationships across cell values. This new version of the table is then used to identify again an evidence set e', which leads to a textual hypothesis that does not reflect the semantics of the original (clean) table.

3 DATA EVIDENCE GENERATION

We distinguish *cold start*, where training data for the target application are not available, and *warm start*, where examples exist.

Cold start. Given a table c from the corpus C, a method to gather evidence is to *randomly* selects subset of cell values from c. In a simple setting, the number of cells for each data evidence e_i can be picked from a uniform distribution between 1 and m, where m in TNLI datasets is usually 10 or less. This method can be extended by profiling any available training corpus for TNLI and obtain a distribution of the evidence size, or this can be provided by the user. For Table 1, a possible data evidence is the set of cells "Mike", "19', "DMBS". Intuitively, with a large number of samples, the random selection eventually models all possible patterns in the tables.

Warm start. If there exist examples T for the application A, we can replace the random selection with a method designed for using T. The intuition is that every example in T has a data evidence e_i that represents a human-defined pattern over the data. Our assumption is that humans express more meaningful patterns that those that we can guess at random. Therefore, being able to capture these patterns enables us to quickly create diverse sets of data evidence.

To identify a pattern, we resort to the task of query synthesis from the cell values in the data evidence. Given an existing example from D, we refer to it as the *seed* s. An example comes with its label l_s, evidence set e_s, textual hypothesis h_s, and the table used to verify it c_s. Given the set of cell values e_s and table c_s as input, we want to identify the query q that outputs such e_s among its results. Executing such query over the original table c_s, we obtain more data evidences $e_1, \ldots, e_n$ that follow the original data pattern in e_s.

Consider again the example in Table 1 with cell values in bold in the first two rows (t_1 and t_2) as seed data evidence e_s. Given such input, we want an algorithm producing a query that returns all pairs of distinct names with their different ages, such as

```
q: SELECT c1.Name, c2.Name as Name2, c1.Age, c2.Age as Age2
    FROM people c1, people c2
```

```
WHERE c1.Age > c2.Age AND c1.Name <> c2.Name
```

Table 2: Evidence cell values sets identified by querying c_s with q derived from e_s.

tid	Name	Name2	Age	Age2
e_1	Mike	Anne	47	22
e_2	Mike	John	47	19
e_3	Mike	Paul	47	18
e_4	Anne	John	22	19
...	...	...	...	..

Query q executed on the seed table c_s returns the relation in Table 2. Each row in the query result is a set of cells for data evidence with the same pattern modeled by the original evidence e_s in the seed. Notice how e_s is also among the results (e_1). Every row in the result of the query can be used to create a new Supports example.

From Examples to Queries. For a given seed example, the textual hypothesis h_s is available. As there are several approaches to infer SQL queries from text (i.e., *text to SQL problem*, or *semantic parsing*), it seems natural to apply one of those to obtain the query above. However, in our approach we derive the query from the data evidence because text to SQL methods are not applicable to our setting. There are three reasons to explain this failure.

First, in a text to SQL task, the input is a NL question and the goal is to obtain the corresponding query. However, our hypothesis are factual expressions. This breaks the assumption in such methods, which are trained on questions such as "What is the region for Kabul?" or "What are the cities in Germany with more than 10000 residents?". Also, when compared to examples in text to SQL corpora, TNLI examples have longer sentences (average of 25 vs 12 words) and contain a larger number of entities (average of 10.5 vs 4.3 nouns). We tested a system [48] over our hypothesis and, in a manual evaluation of its output for 40 hypothesis, it was able to return partially correct SQL queries only in 20% of the cases.

Second, the table structure in semantic parsing datasets is relational only, while TNLI corpora include entity tables, which have attribute labels in the first attribute and are popular on the Web.

Third, even if existing systems could express a query to identify the data evidence e_s precisely, that would not be useful for our setting. For the goal of generating more examples, we need a query that returns the original data evidence e_s as one of its row results, as in Table 2 (tuple e_1), together with more cell sets (tuple $e_2, e_3, \ldots$). The other rows are crucial in our setting as they have the information to produce new examples that follow the same data pattern from the seed. More precisely, given a seed example involving data evidence e_s and table c_s, we are interested in obtaining an *evidence query* (or e-query) q_e that returns the cell values in e_s *in one row of its results* when executed over c_s. This problem is clearly different from general semantic parsing.

While we cannot build on existing solutions, we have the ability to access the data evidence e_s in the seed example, which has precise information about the output of the query. We discuss next how to exploit such seed data evidence to obtain the evidence query.

E-Query Generation. At the core of our solution, we rely on an *evidence graph* to represent relationships among cells in data evidence e_s. Each node corresponds to a cell from e_s and has a label with a pair of values: its attribute label and its row id. The pair acts as identifier for the cell and allows the reconstruction of row and attribute relationships. A (directed) edge across two nodes represent the relationship between their values, e.g., equality, difference, greater than/less than. An example graph derived from data evidence e_s with tuples (Mike, 47), (Anne, 22) is reported in the left hand side of Figure 3, while the right hand side reports a graph for data evidence e_s' with tuples (Anne, NY), (John, NY), (Paul, NY).

Once the evidence graph is derived, we construct the query from it by associating every tuple id across the nodes in the graph to a tuple variable in the query, e.g., e_s leads to two variables in the query, while e_s' to three. These variable are used in the FROM clause, e.g., for e_s we get `FROM people c1, people c2`. We then create the SELECT clause going over the union of the nodes and reporting each node with its variable and attribute, e.g., for e_s we get `SELECT c1.Name, c1.Age, ...`. Finally, we add the conditions by navigating the edges according to their direction (equality, greater than, lower than) and corresponding variable and add those to the WHERE clause, e.g., for e_s we get `c1.Age > c2.Age AND c1.Name <> c2.Name AND c1.Name <> c2.Age AND ...`.

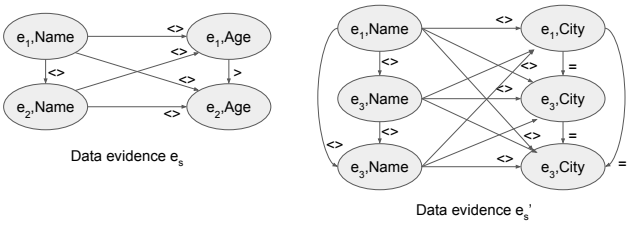

Figure 3: Evidence graphs derived from two seed examples.

Algorithm 1: Generate Query

Input: table c_s , evidence graph g
Output: query q

```
q_s="SELECT"
q_f="FROM"
q_w="WHERE"
visited=[] //Set of visited edges for graph traversal
foreach node n ∈ g do
    tupleIdx = n.getTupleId //Return the tuple idx for the node
    attName = n.getName //Get the attribute name for the node
    alias = 'c' + tupleIdx //Get the relation alias for the node
    if alias ∉ q_f then
        q_f += c_s + ' AS ' + alias +',' //new relation in From
    q_s += alias + '.' + attName + 'AS' + attName + tupleIdx + ',' //new
      attribute in Select
    foreach edge e : (n, d) do
        if e ∉ visited then
            visited += e //add edge to visited
            condition = e.getCondition //Get the condition (<, >, =, or <>)
              for the two nodes
            q_w += alias + '.' + attName + condition + 'c' + d.getTupleID + '.'
              + d.getName + ' AND ' //new condition in Where
q = q_s + q_f + q_w //union of clauses and removal of pending ',/AND'
return q //return query
```

The procedure for query generation is detailed in Algorithm 1. Consider the graph derived from data evidence example e_s in Figure 3. This graph g, together with the table c_s, is our input for the generation of the e-query q. We initialize the three clauses q_s, q_f, q_w of the query with keywords 'SELECT', 'FROM' and 'WHERE', respectively (lines 1-3 in the algorithm). We start the graph traversal from a node n (line 5), for example node with label e_1.Name. We collect the tuple idx for the node (1 in the example, line 6), the attribute name ('Name', line 7) and the relation alias in the query (c1, line 8). The alias for node e_1.Name is not in q_f, so we add it (lines 9-10). We also add the selection condition 'c1.Name,' - pending commas are removed at the end (line 17). We now process outgoing edges for the node, that become conditions in the Where clause (line 12). If an edge has not been visited, we add the corresponding condition to the Where clause (line 16). For example, for the edge going from Node e_1.Name to Node e_2.Name, we add "c1.Name <> c2.Name AND" - pending AND are removed at the end (line 17). The resulting query is obtained by concatenating the three clauses and returning it (lines 17-18). Once a query is derived, its execution on c gives a result table like the one in Table 2.

4 HYPOTHESIS GENERATION

One problem in example generation is converting to NL text the data evidence from a table. This generation process is known as the *data-to-text* problem: given the data evidence, i.e. a set of cells, the goal is to create a sentence for the given cell values faithfully. Existing solutions handle this problem with the creation of *surface sentences*, e.g., for evidence (Mike, 47), (Anne, 22) they describe the cells with a sentence like "Mike has 47 years and Anne 22.".

However, TNLI corpora contain sentences that go beyond surface sentences. Our goal is to generate a variety of hypotheses from the data evidence in the way they are described. For the evidence example above, our goal is to generate also sentences such as "Mike is older than Anne." or "Mike is the oldest person, followed by Anne.". These more challenging hypotheses can still be verified with the same evidence, but require more reasoning and are therefore valuable as training examples for the TNLI models.

To tackle this problem, we resort again to the expressive power of SQL. We split the generation process into two steps: *i*) we compute all the queries over the table such that every query gives as result the data evidence, and *ii*) for every query, we use a PLM M to generate the desired hypothesis. We discuss these two steps next.

4.1 Semantic Queries for Text Variety

Our intuition is that data evidence can be described by the several SQL queries that identify it in the table. These queries are alternative ways to describe the data. By computing the queries, we immediately obtain a *semantic* characterization that can be used to generate hypothesis beyond surface sentences. Given a table c and data evidence e, a *semantic query* (or s-query) over c returns exactly e before the execution of the aggregate functions. Notice that in this case we are not after multiple results, as in the e-query that identifies several examples at the extensional level (over the tuples). The goal is query diversity for the same set of cells; we want variety at the intensional level (over the data description).

Analyzing the examples in popular TNLI corpora [2, 11, 19], we identify two main types of queries.

Table 3: Data evidence e_2 (in bold) and e_3 (underlined).

	Name	Age	City	Team	Salary
t_1	Mike	**47**	SF	DBMS	50k
t_2	Anne	**22**	NY	AI	50k
t_3	John	**19**	NY	DBMS	35k
t_4	Paul	**18**	NY	UOL	55k

Local s-query. This type of query leads to hypotheses related only to the values in the evidence. We name them *local*, as they do not involve information outside the cells in the data evidence. Consider evidence e_1 Table 1; possible queries for it are:

- Surface (or Lookup) s-query: a query that selects cells only with constant selections; `SELECT c1.Name, c2.Name, c1.Age, c2.Age FROM People c1, People c2 WHERE c1.Name = 'Mike' AND c2.Name = 'Anne' AND c1.Age = 47 AND c2.Age = 22`
- Comparison s-query: a query that compares two or more rows by at least one attribute; `SELECT c1.Name, c2.Name, c1.Age, c2.Age FROM People c1, People c2 WHERE c1.Name = 'Mike' AND c2.Name = 'Anne' AND c1.Age > c2.Age`

Global s-query. This type of query generates hypotheses related to information in the entire table. Here, SQL constructs involve constants and attributes not only in the data evidence. If we also consider the table then more queries can be defined:

- Filter s-query: it selects the cells in the evidence according to conditions. For example, given e_1 and Table 1, an s-query that identifies the people with Age greater than 19; `SELECT c1.Name, c1.Age, FROM People c1 WHERE c1.Age > 19`.
- Aggregate s-query: it selects the cells used in an aggregate operation. If the column is numerical, the aggregate function can be sum, avg, count, max, or min. If the column is categorical, then only count is used. Evidence e_1 cannot be identified with an aggregate s-query. However, if the evidence is the entire Age column, as for e_2 in Table 3, an aggregate s-query identifies such values, as we test the exact containment of the cell values involved in the query before the aggregate function. E.g., an aggregate query that returns the highest age in the group is: `SELECT MAX(Age) FROM People`
- FilterAggregate s-query: it selects the result of an aggregate over a group identified by a selection. Evidence e_3 in Table 3 contains cells (22, NY, 50k), (19, NY, 35k), (18, NY, 55k) and can be identified by a FilterAggregate s-query stating that there are three people from NY with an average age of 19.7 years and the minimum salary is 35k: `SELECT COUNT(City), AVG(Age), MIN(Salary) FROM People WHERE City='NY'`

Generating s-queries. Given as input the evidence e and the table c, we want to infer every query q_i such that $q_i(c) = e$ before execution of the aggregates.

Unfortunately, the problem of synthesizing even simple queries from a subset of cells has been shown to be not tractable [43, 52]. Our case is even more challenging as we are interested in queries

with aggregates and filters. To keep the query generation lightweight, our trade-off is to consider only the subset of 5 possible s-query types presented above, effectively biasing the query generation presented in Algorithm 2.

Algorithm 2: Generate S-Query

Input: set of evidence cells e (i.e., a set of $tid.attr$), table c
Output: sQueries

1 *sQueries* = [] //Output
2 *eAttrs* = {} //Set of attributes used in e
3 *eTids* = {} //Set of tids used in e
4 *eAttrsTids* = {} //Dictionary<attr,tids>: ∀ attr. ⟶ set of tids ∈ e
5 *eAttrsVals* = {} //Dictionary<attr,values>: ∀ attr. ⟶ set of values ∈ e
6 *oAttrsTids* = {} //Dictionary<attr,tids>: ∀ attr. ⟶ set of tids ∉ e
7 *oAttrsVals* = {} //Dictionary<attr,values>: ∀ attr. ⟶ set of values ∉ e
8 **foreach** *cell* $v \in d$ **do**
9 *eAttrs* += v.attr; *eTids* += v.tid; *eAttrsTids[v.Attr]* += v.tid
10 *eAttrsVals[v.Attr]* += getCellValue(c, v)
11 **foreach** *cell* $v \in (data \setminus e)$ **do**
12 *oAttrsTids[v.Attr]* += v.tid; *oAttrsVals[v.Attr]* += getCellValue(c, v)
13 n = len(eAttrs) //Number of different attributes in e
14 sQueries += new Surface(project: cells in e)
15 **if** *NOT (sameTIDs(eAttrs, eAttrsTids) ∧ len(eAttrsTids[eAttrs[0]]) > 1)* **then**
16 //If for some attribute there are different seleted tuple ids, only a surface query is allowed
17 **return** sQueries
18 **foreach** $attr \in eAttrsVals$ **do**
19 *comps* = findAllowedOperators($attr$, $eAttrsValues$)
20 **foreach** $comp \in comps$ **do**
21 sQueries += new Comparison(project: cells in e, condition: tid in eTids AND generateBooleanComparisons($comp$, $attr$, e)
22 **if** *eAttrsVals[attr]* ∩ *oAttrsVals[attr]* = ∅ **then**
23 sQueries += new Filter(project: eAttrs, condition: attr in eAttrsVals[attr])
24 sQueries += combineAggregateOperators(aggregate: eAttrs, condition: attr ∈ eAttrsVals[attr])
25 **if** *isNumerical(attr) ∧ min(eAttrsVals[attr]) > max(oAttrsVals[attr])* **then**
26 sQueries += new Filter(project: eAttrs, condition: attr > max(oAttrsVals[attr]))
27 sQueries += combineAggregateOperators(aggregate: eAttrs, condition: attr > max(oAttrsVals[attr])
28 **if** *isNumerical(attr) ∧ max(eAttrsVals[attr]) < min(oAttrsVals[attr])* **then**
29 sQueries += new Filter(project: eAttrs, condition: attr < min(oAttrsVals[attr]))
30 sQueries += combineAggregateOperators(aggregate: eAttrs, condition: attr < min(oAttrsVals[attr])
31 **if** *len(oAttrsTids[0]) == 0 ∧ len(oAttrsTids[i]) == 0 ∧ len(oAttrsTids[n]) == 0* **then**
32 sQueries += combineAggregateOperators(aggregate: eAttrs)
33 **return** sQueries

We first initialize data structures (lines 1-12). *sQueries* contains the s-queries for the given evidence e and table c. For each attribute in e, we keep track of the rows in the evidence (*eAttrsTids*), the cell values of the evidence (*eAttrsVals*), the rows of the data not in the evidence (*oAttrsTids*) and, the cell values in the data not in the evidence (*oAttrsVals*). Considering evidence e_3 and table c in Figure 3, *eAttrsTids[Age]* contains t_2, t_3 and t_4, while *oAttrsTids[Age]* has the only row not included in e_3 for attribute *Age*, namely t_1. Similarly, *eAttrsValues[Age]* contains the three selected values for age, 22, 19 and 19, while *oAttrsValues[Age]* has 47.

Algorithm 3: combineAggregateOperators

Input: attributes $attrs$, possible empty $condition$, evidence e
Output: sQueries $sQueries$

1 *sQueries* = []; *aggrAttrs* = {}
2 **foreach** $attr \in attrs$ **do**
3 *aggrAttrs*[attr] = *findAllowedAggr*($attr$, e)
4 **foreach** *permutation p in permutations(aggrAttrs)* **do**
5 // p contains a list of aggr functions on different attributes *attrs*
6 *sQueries* += new Aggregate(aggregate:p, condition: $condition$)
7 **return** *sQueries*

The data structures are used to check what s-queries can be generated for the input at hand. Surface s-queries can always be generated (line 14), corresponding to returns exactly the cells in the evidence. These queries are flexible and allow us to handle any kind of evidence, while other s-queries require more structured evidence. In particular, to generate comparison, filter, and aggregate queries, all the rows in the evidence should have the exact same attributes selected. In the case when only one row is selected, or some rows have different attributes, the algorithm will stop and only the surface s-query is returned (lines 15-17). To give an example, using evidence with (Mike, 47), (Paul, NY), (DBMS, 35k), the algorithm will generate only the surface s-query.

If the check at line 15 is passed (i.e. we did not interrupt the procedure), we can produce different s-queries. For each attribute in the evidence e, we first check if the attribute enables a comparison among its values in e with the auxiliary function findAllowedOperators (line 19). If some comparison operators are discovered (like <, >, =) then for each comparison (*comp*), we add a new comparison s-query to *sQueries* (line 21). Since in a SQL query in the WHERE clause we can only define pairwise comparisons, we use the utility function *generateBooleanComparisons* to generate all such pairwise comparisons depending on the number of rows in e for the given attribute $attr$. This allows us to generate a WHERE clause that involves t_1.Attr *comp* t_2.Attr … t_{n-1}.Attr *comp* t_n.Attr.

The next s-queries require a filter over one attribute. Such a query can be generated whether a group of values is selected together for an attribute. More formally, we check if, for an attribute $attr$, none of the values selected in e are present in $d \setminus e$ (lines 22-23). In our example e_3, for attribute $City$, the evidence contains the value NY, and it can be considered as a filter since all rows with the NY value for $City$ are selected. The corresponding WHERE clause is $City$ = "NY". Another filter is for a numerical attribute can be triggered when values in the evidence are below/above a constant, e.g., attribute *Age* in e_3, containing all people younger than 47. This check (lines 25 and 28) verifies that all the values in the evidence are greater (lower) than the values for the attribute outside the evidence ($d \setminus e$).

In addition, once a Filter s-query is generated, we check if an aggregate operation can be used to combine them in a FilterAggregate query (lines 24, 27, 30). An additional function, *combineAggregateOperators* performs this check.

Notice that *combineAggregateOperators* generates all the permutations for the allowed aggregate operations on given attributes and generates a set of s-queries (Algorithm 3). In particular, for each attribute *attr*, we first compute the allowed aggregate operations (line 3). Given an attribute, *findAllowedAggr* returns count() for categorical attributes and count(), avg() for numerical attributes; if e contains also the min/max value in d then it also returns the min()/max() function. We then generate all the permutations of the attributes and the aggregate operation for each attribute. For example, evidence e_3 in Table 3 admits a count(), avg(), and min() for the Age and Salary attributes, while allowing only count() for City. Thus possible permutations generated include [count(Age), count(City), count(Salary)], [avg(Age), count(City), count(Salary)], [min(Age), count(City), avg(Salary)].

Finally, in lines 31-32 of Algorithm 2, we check if an entire column is selected, and for them, we generate multiple Aggregate s-queries with the same approach described above.

We are not claiming that the list of s-query types above is exhaustive and covers all the possible queries that identify the evidence. For example, we are not covering the "Order by" s-query, e.g., the one for e_1 that identifies the top 2 people with the highest Age: `SELECT Name, Age FROM People c1 ORDER BY Age DESC LIMIT 2`, leading to the sentence "Mike and Anne are the two oldest persons.", or "Group by" s-query, where one might want to compare aggregate values between two groups to generate a sentence like "People in DBMS team have an average salary higher than people from AI team.". However, our study of the TNLI corpora shows that the five types above cover most of the hypotheses used in practice. In Section 6.3, we show how additional s-queries have a small positive impact in the accuracy on the target TNLI application.

Table 4: S-Queries generated by Tenet.

Name	Example
Surface	**SELECT** a1.Name, a2.Name, a1.Age, a2.Age **FROM** People a1, People a2 **WHERE** a1.tid = t_1 *AND* a2.tid = t_2
Comparison (<, >, =)	**SELECT** a1.Name, a2.Name, a1.Age, a2.Age **FROM** People a1, People a2 **WHERE** a1.tid = t_1 *AND* a2.tid = t_2 *AND* a1.Age > a2.Age
Filter	**SELECT** Name, City **FROM** People **WHERE** City in "NY"
FilterAggregate	**SELECT** max(Age) **FROM** People **WHERE** City in "NY"
Aggregate	**SELECT** count(Name), avg(Age) **FROM** People

Table 4 reports the different type of s-queries that might be discovered from Algorithm 2, together with some examples.

4.2 Text Generation

Once we know for each evidence the possible s-queries, we generate textual sentences that form the hypothesis for the TNLI example.

We exploit the text generation capabilities of pre-trained large language models (PLMs), such as those in the GPT family [8]. A PLM is trained over huge amounts of textual data, which gives it proficiency in writing, and on source code, which gives it the ability to be instructed with functions.

Table 5: Functions used by Tenet in ChatGPT prompts.

S-Query	Function	Example
Surface	read(attrList)[*]	Anne is 22 years old and Paul is 18.
Comparison	compare(op, attr)	Anne is older than Paul.
Filter	filter(cond, attr)	Anne, John and Paul are from NY.
FilterAggregate	filter(cond, attr); compute(func, attr)=val	The oldest person from NY is 22 years old.
Aggregate	compute(func, attr)=val	Mike is the oldest person.

For each s-query, we define a task that describes the text generation function that we want to use. Such generation functions are defined by us with the prompts for the PLM. The text generation functions mapped to the relative s-queries are reported in Table 5 with examples of the text they generate. The *op* parameter is related to operators =, < and > for numerical attributes, while = only for categorical attributes. The *cond* parameter is the condition used in the WHERE clause. The *func* parameter refers to an aggregation function among count, avg, sum, min, or max. For example, given the Filter s-query `SELECT Name, City FROM People WHERE City in ("NY")`, we derive the operation `filter(in ("NY"), City)`.

To avoid hallucination of the model in calculating aggregate functions, for `compute` functions we calculate the value *val* from the evidence and feed it to the PLM; this enables us to avoid such computation with the PLM, as it brittle to this task. For example, given the evidence e_3 in Table 3, we explicitly calculate the avg for the attribute Age, and use the calculated value in the operation *compute(avg,Age)=19.66*. This helps the PLM to express a sentence like "The average age is 19.66". In general, our approach based on evidence and s-queries returns factual hypothesis, while a baseline solution based only on prompts for the PLM create examples with hallucination that ultimately lead to worse performance in the target TNLI task. We report on this comparison in Section 6.3.

Since in most cases PLM cannot be fine-tuned, as they are offered with APIs, we opt for using them with a prompt with some examples (few shots, or in-context learning in NLP terminology) [12]. The prompt consists of two parts.

The first part is fixed and contains 16 examples of how to verbalize the data evidence with an operation. For every example, it reports the table name and the text linearization of the schema [4]. Then each row of the evidence is linearized in the same way. If some attribute lack a value in the evidence, we add "null" as the cell value. Finally, we define the expected textual hypothesis. For example, a comparison operation is reported in the left hand side of Figure 4 for a data evidence with two tuples and two attributes.

Table: Cars
Model | Year
———————
500 v3 | 2012
Clio v6 | 2018
———————

Function:
`compare(>,Year)`
Example:
"The 500 v3 is an older model than the Clio v6"

Table: Table Name
$Attr_1$ | $Attr_2$ | ... | $Attr_n$
———————
v_1^1 | v_2^1 | ... | v_n^1
...
v_1^m | v_2^m | ... | v_n^m
———————

Function: *(Desired verbalization expressed by function f)*
Example:
(it returns a textual sentence using v_1^1 ... v_n^m according to f)

Figure 4: One of the 16 examples for in-context learning (left) and generic serialization of the evidence in the prompt at test time (right).

The second part reports unseen data evidence and an operation to steer the model to generate the desired text. The right hand side of Figure 4 shows the input before instantiating it with the table and operation at hand.

5 REFUTES EXAMPLES GENERATION

The methods above produce Supports examples, i.e., the label states that the data evidence entails the textual hypothesis. This is true by construction, as the hypothesis are derived directly from the evidence. However, TNLI applications have also Refutes examples, where the evidence contradicts the hypothesis. Our approach to the generation of Refutes examples relies on our method for generating the Supports ones. We generate a Refutes example for every Supports one. Given some evidence e from the original input table c, we inject noise in a copy c', so that we derive a new evidence e'. An hypothesis h' is then derived from e'. Hypothesis h' is a Supports sentence for c', with evidence e', but it is also a Refutes sentence w.r.t. the original (clean) table c and evidence e. The new example is the tuple with label Refutes, c, h' and evidence e.

Table 6: A modified version of the "People" table with shuffling of the original "Age" values and one injected tuple.

	Name	Age	City	Team
t'_1	Mike	18	NY	DBMS
t'_2	Anne	19	NY	AI
t'_3	John	22	SF	DBMS
t'_4	Paul	47	NY	UOL
t'_5	Mary	17	NY	SYS

Consider again Table 1, denoted as c and the evidence e=(Mike, 47), (Anne, 22). First, we create a copy c' of the table and manipulate it to inject noise. We shuffle in c' the values for 50% of the attributes (we discuss in Section 6.3 how we set this threshold) involved in e. The resulting table is reported in tuples t'_1 - t'_4 in Table 6, only Age has been shuffled. This step breaks the original relationships across cell values at the tuple level. We then either introduce a new tuple in c', such as t'_5, or remove from c' one tuple at random. This step changes the cardinality of the tuples, which is key for s-queries involving aggregates, and introduces out-of-domain values. The generation of the new values depends on the type. For categorical attributes, we use a PLM, which generates "Mary", "NY" and "SYS" for tuple t'_5. For numerical attributes, we generate lower/higher values than the min/max value for every active domain - these new values break the original min/max/avg property for the updated attribute, e.g., the new min value "17" in t'_5. Finally, we remove from c' any row that appears in c.

Given the new "noisy" table c', we directly apply the generation of Supports claims from Section 4. We use evidence e to generate an e-query q over c, then we use q to obtain the new evidence e'. Finally, we generate h' from e' and c'. Hypothesis h' is supported by evidence e' (table c'), but it is refuted by original evidence e (table c). For example, a claim may state "Mike is younger than Anne", which is refuted as hypothesis w.r.t. the data in Table 1. As another example, consider the evidence e_2 for the FilterAggregate s-query (Table 3), which takes all Age values. In this case, there is no shuffling, but the new evidence e'_2 includes 17. Therefore a Refutes claims for operation Max cannot be generated, as (47) is a valid evidence in e_2, but an hypothesis involving Min can be generated, as (17) is not in e_2.

6 EXPERIMENTS

We organize our evaluation around four main questions. First, does Tenet automatically generate training data of quality comparable to those manually created by human annotators? Second, what is the impact of the information stored in the PLMs at the core of most inference models for TNLI? Third, what is the impact of the models and parameters used in Tenet? Fourth, what are the costs of Tenet, in terms of execution time and budget for external APIs?

Before getting into the discussion of the results, we present datasets, models, and metrics used in the evaluation.

Table 7: Statistics for the datasets. All datasets except OutOfDomain and Swapped have train and test splits.

		Source	# of examples	Avg hyp. length	Avg # of row/atts
Train	Feverous	Wiki	10k	122.0	10/3
	TabFact	Wiki	92k	73.0	14.5/6
	Infotabs	Wiki	16.5k	55.5	14.5/2
Test	Feverous	Wiki	1k	123.2	10/3
	TabFact	Wiki	25k	70.8	14.5/6
	Infotabs	Wiki	7k	57.1	15.5/2
	OutOfDomain	UCI	0.15k	45.0	16/8
	Swapped	¬(Wiki)	1k	105.0	10/4

Train Datasets. We use three datasets from the TNLI literature: Feverous [2], TabFact [11], and Infotabs [19]. Each dataset comes with one subset (split) of examples for training and one for test. Every annotated example consists of a *table*, a *textual hypothesis*, *data evidence* (subset of the table), and a Supports/Refutes *label*. All examples are manually written by humans. We report dataset statistics in Table 7; "Avg # of row/attributes" is per table.

As a baseline, we extend the original training datasets with an augmentation for text [15]. Given an example, we produce seven new versions of it by changing the textual hypothesis using back

translation, wordnet, word2vec, synonyms, random word swap, random word deletion, random word insertion (*Aug*).

We also produce training datasets for our techniques. Given a corpus of tables, we always generate the *Tenet Cold* (*TenetC*) dataset (Section 3). If examples have annotation for data evidence, we can also generate the dataset for *Tenet Warm* (*TenetW*). Hypothesis are created with s-queries (Section 4) and negative examples are generated according to Section 5. For each given table, we produce three Supports and three Refutes hypothesis, therefore all TENET datasets are balanced in terms of labels.

For every table, TENET creates one example with a surface query (cause those are the most popular kind in the corpora and can always be generated) and two for the two rarest s-queries among the other four types (Comparison, Filter, Aggregate, FilterAggregate). Table 4 reports s-queries from the more commonly observed in the corpora to the rarer. If the complex s-queries cannot be generated, the remaining examples are obtained with surface queries.

Test Datasets. The datasets from previous papers (FEVEROUS, TABFACT, and INFOTABS) have their own testing datasets with annotated examples manually written by humans (statistics in Table 7). However, as all these models use tables from Wikipedia, we create also a test dataset with eight out-of-Wikipedia (OUTOFDOMAIN) tables selected from different sources. Finally, TENET can go beyond its role in the training step and be used also to generate test datasets, which is useful for evaluation of existing methods. In this spirit, we also generate a test dataset, SWAPPED, as described in Section 6.2.

Inference Models for TNLI. In this work, our goal is to show the quality of automatically generated training data. We therefore do not propose new TNLI models and adopt the ones in the original papers. In FEVEROUS, the inference predictor is a RoBERTa (large) encoder fine-tuned for classification on multiple NLI datasets [31]. In TABFACT, the inference predictor is built as a program synthesis problem, modeled as a latent program search followed by a discriminator ranking [29]. In INFOTABS, the inference predictor is also a RoBERTa (large) encoder fine-tuned for classification.

Pre-trained Language Models. For the hypothesis generation (Section 4) and the error injection (Section 5), we assume that a pre-trained language model (PLM) is available. We tested several PLMs and use ChatGPT as default. We report a comparison of T5, fine-tuned on ToTTo, and ChatGPT in Section 6.3.

Metrics. We report accuracy for the TNLI task: how many Supports/Refutes classification decisions are correct over the total number of tests. We also report execution times and cost (for external APIs) in running the models (Section 6.4).

6.1 Quality of Training Examples

We start by comparing results with training data with examples generated from the same sets of tables. The tables are taken from FEVEROUS, TABFACT, and INFOTABS datasets. As state of the art solutions, we directly use the manually written examples (*Human*), eventually augmenting them (*Human+Aug*). For TENET methods, we take the corresponding tables of the original training data and generate examples with *TenetC* and *TenetW*. For every experiment, we increase the number of input tables, collect or generate the examples, and run the inference model to compute the accuracy

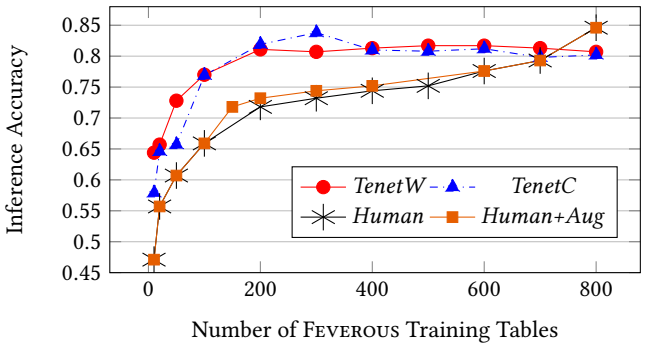

Figure 5: Inference accuracy for different training datasets over the FEVEROUS test data. The *x* axis is the number of tables in training set. *Human* is FEVEROUS original training data.

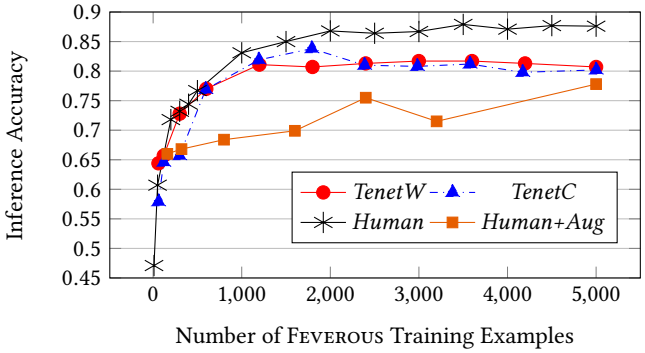

Figure 6: Inference accuracy for different training datasets over the FEVEROUS test data. The *x* axis is the number of examples in training set. *Human* is the FEVEROUS training data.

on the test data. For example, given a subset of the original examples in FEVEROUS training corpus, *TenetC* generates evidence and hypothesis using only the table in every example, while we use the original example for *Human*. We finally assess the quality of the examples, both original and generated, on the same test splits.

The TNLI accuracy results in Figure 5 for the FEVEROUS test data show the impact of examples, which is a proxy for their quality. Up to 700 input tables, both TENET-generated datasets outperform the examples written by humans, with more than 20 absolute points in cases with less than 150 tables. Even with only 200 tables available for the training step, both TENET example generation methods achieve an accuracy over 0.8 on the (manually crafted) original test data. If we augment the Human examples with those generated by *TenetW*, we observe accuracy at 0.8 even with only 150 tables in the training corpus.

TENET benefits by the fact that for every input table, it extracts one data evidence and generates three Supports and three Refutes examples, while the humans wrote one example per table. To make a comparison over the same number of examples, we report the same experiment, but with results plotted according to the total number of examples, regardless of the number of tables.

Figure 6 compares the results obtained with sets of examples of the same size, but from different methods, on the FEVEROUS

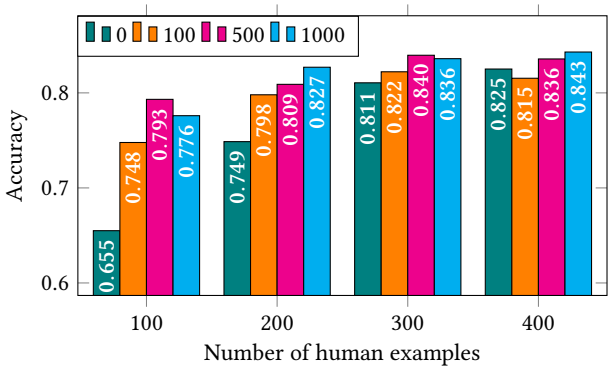

Figure 7: Inference accuracy on FEVEROUS when training with the union of human examples (100 to 400) and TENET generated examples (0 to 1000). The first bar is for *Human* examples only, other bars are for *Human+Tenet* examples.

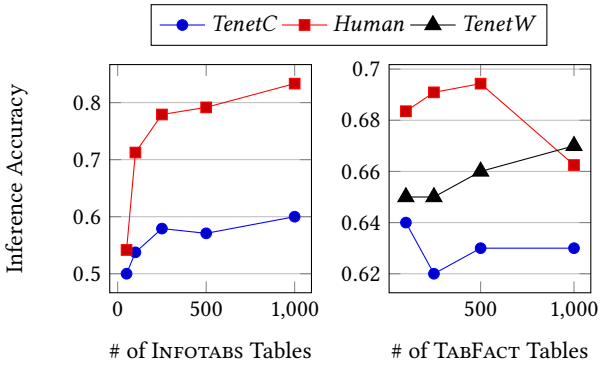

Figure 8: Inference accuracy for different training datasets over INFOTABS (left) and TABFACT (right) test data. The *x* axis is the number of tables in training datasets.

test data. TENET's examples (both *TenetW* and *TenetC*) lead always to higher accuracy than the original examples with traditional augmentation (*Human+Aug*). Moreover, TENET's examples lead to comparable accuracy w.r.t. the human-written corpus up to around 1.5k examples. After this value, the results are quite stable for our generated datasets, while they slowly increase for those written by humans. This is consistent with Figure 5, as *Human* outperforms TENET when using at least 800 tables. Our explanation is that there is a long tail of reasoning cases that are not covered by the five s-queries that we have designed, e.g., FEVEROUS test data has a small fraction of hypothesis involving arithmetic operations. While new s-queries can be added, the plot shows that with only five types we can already obtain automatically very good training datasets.

Figure 7 reports the results for the training done with a combination of *Human* and *Tenet* examples for FEVEROUS. We report the impact of different numbers of generated examples. Increasing the size of the generated training data increases the accuracy on the test set. The benefit of TENET examples is higher with smaller numbers of human training examples.

Table 8: Accuracy on FEVEROUS test set augmenting the original train set with TENET and text augmentation examples. TENET-X stands for examples generated from X tables.

Train set	Augmented	Augmented Size	Accuracy
FEVEROUS	-	-	0.909
FEVEROUS	TENET-50	153	0.910
FEVEROUS	TENET-100	321	0.916
FEVEROUS	TENET-200	683	0.911
FEVEROUS	TENET-300	**1018**	**0.917**
FEVEROUS	TENET-400	1357	0.910

Table 8 reports results for a combination of *Human* and *Tenet* train examples on FEVEROUS. We augment the entire original training set with TENET's examples using an increasing number of seed tables. The best accuracy is obtained with 300 tables (1018 training examples) and an accuracy of 0.917. A larger number of generated examples has a smaller impact. We observe a similar pattern with the baseline text augmentation [15]: adding all augmented examples to the original human examples leads to a lower accuracy (0.908).

Figure 8 reports the results for the accuracy of the inference model for INFOTABS and TABFACT test datasets. For both datasets, the examples generated by human annotators do better than TENET examples. One difference from FEVEROUS is that these datasets have up to eight examples per table, therefore the accuracy grows faster with more tables compared to Figure 5. For TABFACT, the difference is a few points, while for INFOTABS is more significant. This is because the latter contains only entity tables derived from Wikipedia info-boxes. Those are equivalent to tables with a single row and many attributes, thus not suitable for s-query generation and our algorithm defaults to surface hypothesis for these cases. Finally, INFOTABS examples use the whole table as data evidence, which explains why we cannot derive e-queries for *TenetW* for it. However, we remark that our examples are generated without involving humans, therefore with a cost that it is a fraction of the one to obtain the original training datasets.

6.2 Impact of Information from Pre-training

In this experiment we measure the impact of the knowledge stored in PLMs. Are the inference models really using the input evidence and tables? Or they rely on the information in the PLMs?

Indeed, PLMs have been trained with large amount of information, including dump of the Web and Wikipedia. With existing datasets derived from Wikipedia, it is not obvious how much of the inference decision comes from the information gathered in the weights of the large language model and how much is coming from the evidence and table passed as input for the TNLI task.

To enable such analysis, we use two test datasets: OUTOFDOMAIN and SWAPPEDFEVER. We design OUTOFDOMAIN with five tables from the UCI repository [14] (Abalone, Adults, Iris and Mushroom) and three sports tables used in NLP text generation challenges [53]. Hypothesis and labels are manually crafted by the authors with the generation process outlined in the Feverous paper [2].

For SWAPPED, the goal is to create hypothesis that contradict the information in Wikipedia. For this task, we create hypothesis that are supported by the tables given as input, but are in contradiction with the original Wikipedia tables, which are likely present as learnt information in the PLMs. To create this dataset, we take tables *O*

from the Feverous corpus and create Supports hypothesis A with our methods. We then inject errors in the tables, obtain tables O', and create Refutes hypothesis B. We then swap the labels in the examples. We change the labels of the original Supports hypothesis A, as they are now Refutes for tables O', and do the same for B. The (now) Supports hypothesis in examples B are supported by the provided tables, but are in contradiction with the original Wikipedia tables that have been used in the pre-training of the PLMs.

For this experiment, we train the FEVEROUS inference predictor on TENET training data and on the original FEVEROUS datasets as in the previous section.

Table 9: Accuracy results for test datasets OUTOFDOMAIN, derived from non-Wikipedia tables, and SWAPPED, with examples contradicting information in Wikipedia tables. Training examples (5k) derived from FEVEROUS tables.

	Generated Train Set		FEVEROUS Train Set	
Test set	***TenetW***	***TenetC***	***Human***	***Human+Aug***
OUTOFDOMAIN	**0.84**	0.80	0.77	0.76
SWAPPED	**0.65**	0.65	0.64	0.61

The results in Table 9 show two important insights. First, accuracy results are lower compared to the original datasets from the literature. This is because those inference tasks are defined over concepts and entities that are already "known" to the PLMs used in the inference. This is evident with the SWAPPED dataset that contradicts the original knowledge in the Wikipedia tables used in the pre-training of the PLM. Models that rely on the provided data evidence, rather than PLMs' knowledge, are more robust when executed on new domains.

Second, the model trained on TENET data outperforms the models trained with humans' examples. Our examples better steer the inference model into learning to use of the data evidence, rather than the internal information in the PLM. This is especially important for domain-specific tables that are covering entities not on the Web, with an improvement of 7 absolute points with *TenetW*'s model over the humans' model.

6.3 Ablation Study

In this section, we first measure the impact of the PLM on the quality of the generated examples. We then study the impact of parameters used across the data evidence and hypothesis generation.

Role of PLM. As a baseline for the first experiment, we report for the training data produced directly by a pretrained language model for this task (*PLM*). We use ChatGPT to automatically generate hypothesis from tables given only a prompt with the instructions and examples. For each table of a given dataset, ChatGPT generates (i) three Support and three Refuse hypothesis using data in the table, and (ii) the set of cells used to produce each sentence (evidence). Tables are presented using the same linearization of Figure 4.

Table 10 shows that TENET generates valid examples independently from the PLM used in the hypothesis generation. This applies with T5, fine tuned on ToTTo [39], and with ChatGPT, with in-context learning. *PLM* creates useful examples, but without the guide of the data evidence and the s-queries, it is prone to hallucinations that degrade the quality of the training data. In other words, generating examples out of the PLM is doable but TENET methods get higher quality. On average, TENET with ChatGPT has slightly better results because of its superior ability in text generation. However, using OpenAI API comes with its own issues, in terms both of data privacy, usage cost, and execution time (Section 6.4).

Table 10: Accuracy results with different PLMs for example generation. Same inference model trained on examples from 300 tables from FEVEROUS train corpus.

	TenetW Train		*TenetC* Train		*PLM* Train
Test set	**T5**	**ChatGPT**	**T5**	**ChatGPT**	**ChatGPT**
FEVEROUS	0.79	0.80	0.81	**0.82**	0.70
TABFACT	0.60	**0.65**	0.56	0.62	**0.65**
INFOTABS	n.a.	n.a.	0.57	0.51	**0.63**
OUTOFDOMAIN	0.81	**0.84**	0.80	0.80	0.69
SWAPPED	**0.67**	0.65	0.63	0.65	0.58

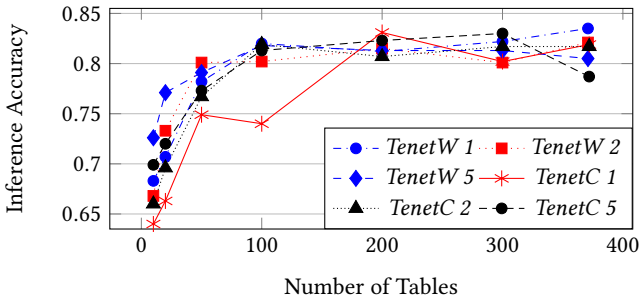

Figure 9: Impact of 1, 2, 5 data evidence per table in example generation. FEVEROUS test data, 3 s-queries per evidence.

Impact of Parameters. Figure 9 shows the inference accuracy when varying the number of results taken from the evidence-query for every table. The experiment is run over the FEVEROUS test data, with tables in its training data, and with TENET models that use three s-query for every data evidence. Results show that a larger number of data evidence per table leads to better results with very few tables, but has marginal gain with an increasing number of tables in the training. We explain this behavior with the fact that using examples from more tables is more beneficial than using the multiple examples from the same table. For a trade off for quality and cost of example generation, we set one data evidence as default.

Figure 10 shows accuracy results when varying the number of s-queries executed for every data evidence. The experiment is over the FEVEROUS test data, with tables in its training data, and with TENET models using one result from the e-query. Results show that more hypothesis lead to better results on average, especially with small numbers of tables. As a trade-off between cost and quality, we set three s-queries as default.

Impact of different thresholds in refute example generation. To identify the right percentage of attributes to shuffle, we test different threshold τ values (Section 5). We use 200 FEVEROUS tables and generate positive and negative examples with TENET. We then

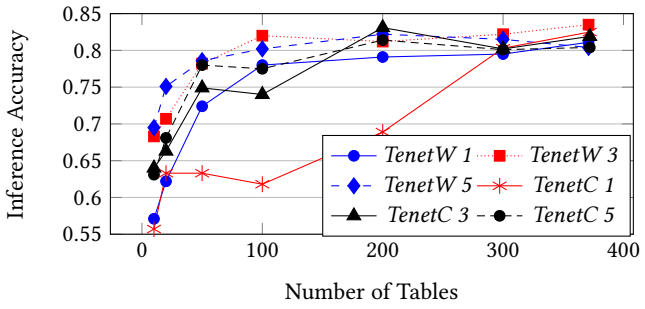

Figure 10: Impact of 1, 3, 5 s-queries per table in example generation. FEVEROUS **test data, 1 data evidence per table.**

Table 11: Accuracy results with different thresholds for the # of shuffled attributes in *Refutes* example generation.

Test set	Type	Quality with $\tau = 0.25$	Quality with $\tau = 0.5$	Quality with $\tau = 0.75$
FEVEROUS	Cold	0.74	**0.77**	0.69
FEVEROUS	Warm	0.74	**0.77**	0.74
INFOTABS	Cold	0.54	**0.57**	0.54
TABFACT	Cold	0.50	**0.62**	0.54
TABFACT	Warm	0.52	**0.65**	0.57

train the model and measure the inference accuracy. Results in Table 11 show that 50% leads to the best quality.

Table 12: Accuracy results with two additional s-queries.

Test set	Type	Standard S-Queries	New S-Queries	Improvement
FEVEROUS	Cold	0.77	0.78	+0.01
FEVEROUS	Warm	0.81	0.83	+0.02
INFOTABS	Cold	0.59	0.59	0
TABFACT	Cold	0.65	0.63	-0.02
TABFACT	Warm	0.65	0.66	+0.01

Impact of new s-queries. To define the impact of adding new s-queries, we extend the set in Table 4 with two new s-queries: *ranked*, which uses the RANK() function in Postgres to craft examples such as "John is the second youngest person", and *percentage*, which calculates the difference in % for pairwise numerical values to generate examples such as "Bob earns a salary that is 50% higher than John's". These kinds of examples are present in a small percentage in TNLI corpora. We use such new s-queries over 200 Tables and extend the original TENET training data with the corresponding training examples. Accuracy results in Table 12 show that adding examples from the two new s-queries brings a small gain in quality.

6.4 Execution Time and Cost

We measure TENET execution time to generate training data. We create five samples of 200 tables from FEVEROUS and execute the full pipeline with Cold and Warm approaches. We report in Figure 11 the average time in generating a single training example. We partition the overall time across the generation of the new evidence (blue, bottom), the hypothesis generation (orange, middle), and the text

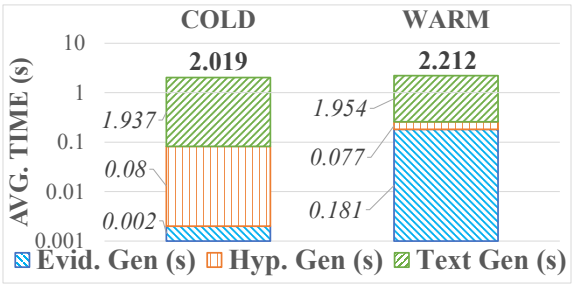

Figure 11: Average time for generating one example with Cold and Warm approaches. For each scenario is reported the time taken by each generation step, with total time on the top of each bar. Time in seconds and reported in log scale.

generation with ChatGPT (green, top). The average time does not change significantly between the cold and warm approaches. In the warm approach, more time is spent in the evidence generation. Indeed, generating and executing the e-queries takes more time than random selection. On the other hand, using a seed evidence in the warm approach leads in most cases to more compact evidence, involving a smaller number of attributes compared to random. The cold setting, due to its random nature, involve several attributes, and thus generates more s-queries to check. The most expensive step in our approach (97% of the execution time) is due to text generation. This heavily depends on the ChatGPT availability and it takes on average from 1.5 to 2.2 seconds per request.

Table 13: Costs of generating hypothesis with ChatGPT.

	# Tables	# Positives	# Negatives	Total #	Price ($)
Warm	200	1670	1536	3206	11.6
Cold	200	1655	1580	3245	11.7

Table 13 reports the costs of generating hypothesis with the OpenAI API and ChatGPT for 200 tables. The cost linearly depends on the number of generated examples, as ChatGPT calculates the costs based on the size of the input prompt together with the size of the generated output. On average the generation of one example costs 0.0037$. The total cost of all the experiments reported in this paper is about $130 for 36K generated examples. Using a smaller PLM, such as T5, on a local machine (Apple M1 Max laptop) does not have any API cost and takes on average 1 sec for the text generation step of one training example. However, the quality of the generated text is slightly lower than ChatGPT.

In conclusion, TENET generates a training example with a lower time and cost w.r.t. those required by human annotators.

7 RELATED WORK

TENET is a system spanning different problems. We start discussing augmentation and generation of text examples. We then focus on extracting SQL statements from NL text and from query results. Next, we cover text generation from tabular data. Finally, we discuss applications in Tabular Natural Language Inference (TNLI).

Augmentation of Textual Examples. In augmentation, the goal is to provide additional annotated data by modifying existing examples. In one baseline, we use a method that augment examples to create more hypothesis for the same tabular data evidence.

Augmentation can be performed on the data or on the feature space [6]. In the data space, several works operate at the *character* level [7, 16] by swapping, removing, adding letters; injecting common spelling mistakes; or replacing words with abbreviations, e.g., "I'm". Approaches that operate at the *word* level, use word swap/deletion [3, 5, 21, 25, 40] or replacement with synonyms, hypernyms, and antonyms [16]. At the *document* level, a popular method is round trip translation [1, 55]. New textual samples are also created with generative methods [40] and pre-trained language models (PLMs) [8, 13]. For example, by using GPT-2, as a generator, and reinforcement learning to guide it towards specific class labels in the decoding stage [30]. We also use PLMs (ChatGPT and T5) for text generation, but our generation is driven by the relational data.

Other works focus on transforming the feature space, rather than the raw data. Noise addition is used to create new examples by modifying the vectors with the injection of zeros [44] or updating them with random multiplications [26]. An alternative to noise is interpolation, such as combining similar vectors from examples with the same label [9, 49]. This line of work is not applicable in our case where the evidence table data is explicit in the example.

Generation of Textual Examples. For example generation in the unsupervised setting (no examples available), several works focus on exploiting PLMs to obtain textual claims. In *SuperGen* [32], an original text t is combined with a template prompt to obtain a positive, neutral or negative sentence from the PLM, e.g., given sentence t, the prompt for a new negative sentence is "t. However the truth is...". In the supervised setting, humans are asked for hints on the output, e.g., by annotating a taxonomy with related words to train a LSTM model that generates sentences [34]. In another direction, the classifier is trained with examples from a fine-tuned text generator [33] or with examples extracted from Wikipedia paragraphs with Bart models [27] that obtain pairs of (claims, label) [38]. While these works share some ideas with our approach, they cannot consume tables as input.

Semantic Parsing. In the supervised setting, we generate data evidence for new samples from a given example. As we want full control on the data (to distinguish Supports and Refutes), we derive a SQL query for every data evidence. Text-to-SQL (semantic parsing) methods that infer the query from the given hypothesis [17, 23, 51]. perform poorly when executed on factual claims. For instance, RAT-SQL [48] derives a query from a textual NL question and table pair. While it handles datasets with multiple tables and foreign keys [57], it assumes relational tables only, works on questions (not factual claims) as input, and mostly returns incorrect queries in our setting.

Query Reverse Engineering. In this problem, the goal is identify the query that generates a given output. Deriving surface-queries, that overfit on the input, is always possible, while for more general queries the complexity is exponential [52]. However, some methods focus on getting one query for the given example [46, 52], in this case the complexity is in P-time under some assumptions. This is not suitable for us, as we want to find a variety of s-queries to reflect the different kinds of reasoning needed in the inference. Moreover, some of these methods require both positive and negative output examples [52], while we have only (positive) data evidence. Related approaches for query-by-example also propose heuristics for the discovery of sets of possible queries, but the solution is for interactive use [28], while in our case we aim at full control over the variety of s-queries. Finally, given the nature of the corpora in NLTI problem, we do not focus on the inference of joins [58].

Text Generation from Tables. There are works on verbalizing tables to produce sentences that describe them. *Data-to-text generation* has been traditionally tackled by leveraging domain knowledge and complex grammar rules [24, 42]. Recent breakthroughs in NLP, remove cumbersome sentence and content planning [39]. *R2D2* [36] combines a generator with a faithfulness discriminator for the produced text t to reduce "hallucinations" such as entities appearing in t that are not in the data. *DocuT5* [45] tackles the lack of context in describing data by manually adding table information and foreign keys. In our setting, we use table names, captions, and any document structural information as context. These works lead to fluent sentences, but only in the form of description of the tuples. In analogy to queries, they describe the output of *look-up* operations. We extend these approaches by generating textual claims that describe data retrieved with SQL operations beyond simple look-up, such as aggregates. *LogicNLG* [10] also discusses the requirement of logical operations in the generated text to go beyond the surface realization of a set of cell values. They create a dataset with more complex examples, such as math operations and comparisons, and test sentence generation with several methods. Our work introduces prompts based on few-shots for generative models, such as GPT-3, which perform better than the previous methods.

Tabular NLI. TNLI determines if a textual hypothesis is supported or refuted based on a given premise in tabular format. Applications include computational fact-checking [35], table understanding [19, 20] and assistance in data-centric fields such as finance and healthcare [37]. As an example of existing datasets, *Feverous* is a collection of labelled textual claims generated by a crowd starting from Wikipedia pages [2]. The pipeline for fact checking is composed of a cell retriever (given the claim and the tables) and a veracity predictor (given the claim and data evidence). Similarly to Feverous, the *SemEval-2021 Task 9* [50] has 2k tables on which claims are built for fact verification and cell evidence selection. From the tables, claims for the training set are generated using IBM Watson Discovery and test claims are written by annotators. The claim generation is based on templates and is poor in terms of variety. In *TabFact* [11], the examples rephrase table data with operation on cells, such as count and max, to obtain the claim. One checking method uses a linearized table with a BERT model, while a second method uses Latent Program Analysis. In *InfoTabs* [19], annotators build a dataset with 3 sentences for each table. They test various pre-trained NLI systems on their dataset and conclude that they do not perform well.

8 CONCLUSIONS

We proposed a generic solution that automatically constructs high-quality annotated datasets for TNLI. Experiments show that given only a table as input, Tenet creates examples that lead to high accuracy when used as training data in the target tasks. Even in

settings with a small number of tables for the training of the system, TENET produces examples with variety both in the pattern of the data and in the reasoning used to verify or refute the hypothesis.

While TENET is an important first step, there are several research directions still open. First, there are classes of examples in the long tail that are not represented in our generation process. Examples include mathematical operations, such as hypothesis "Mike is 27 years older than Anne". As the number of possible s-queries is large, we envision a solution where s-queries are inferred from hypothesis in annotated corpora, similarly to what we do for e-queries, with a new learning task that extends existing work on semantic parsing [23]. Second, existing corpora contain also examples that span multiple tables or even tables and text, but our e-query generation algorithm must be extended for such settings. In a similar direction, new algorithms for e- and s-queries are needed to generate examples than require joint reasoning over text and tabular data [56]. Third, once models have been bootstrapped with TENET, we could design active learning algorithms to solicit human-written examples that effectively improve performance on the test set.

REFERENCES

[1] Milam Aiken and Mina Park. 2010. The efficacy of round-trip translation for MT evaluation. *Translation Journal* 14, 1 (2010), 1–10.

[2] Rami Aly, Zhijiang Guo, Michael Sejr Schlichtkrull, James Thorne, Andreas Vlachos, Christos Christodoulopoulos, Oana Cocarascu, and Arpit Mittal. 2021. FEVEROUS: Fact Extraction and VERification Over Unstructured and Structured information. In *NeurIPS (Datasets and Benchmarks)*.

[3] Ateret Anaby-Tavor, Boaz Carmeli, Esther Goldbraich, Amir Kantor, George Kour, Segev Shlomov, Naama Tepper, and Naama Zwerdling. 2020. Do not have enough data? Deep learning to the rescue!. In *AAAI*, Vol. 34. 7383–7390.

[4] Gilbert Badaro and Paolo Papotti. 2022. Transformers for Tabular Data Representation: A Tutorial on Models and Applications. *Proc. VLDB Endow.* 15, 12 (2022), 3746–3749. https://www.vldb.org/pvldb/vol15/p3746-badaro.pdf

[5] Markus Bayer, Marc-André Kaufhold, Björn Buchhold, Marcel Keller, Jörg Dallmeyer, and Christian Reuter. 2022. Data augmentation in natural language processing: a novel text generation approach for long and short text classifiers. *International Journal of Machine Learning and Cybernetics* (2022), 1–16.

[6] Markus Bayer, Marc-André Kaufhold, and Christian Reuter. 2022. A Survey on Data Augmentation for Text Classification. *Comput. Surveys* (jun 2022).

[7] Yonatan Belinkov and Yonatan Bisk. 2017. Synthetic and natural noise both break neural machine translation. *arXiv preprint arXiv:1711.02173* (2017).

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[9] Jiaao Chen, Zichao Yang, and Diyi Yang. 2020. Mixtext: Linguistically-informed interpolation of hidden space for semi-supervised text classification. *arXiv preprint arXiv:2004.12239* (2020).

[10] Wenhu Chen, Jianshu Chen, Yu Su, Zhiyu Chen, and William Yang Wang. 2020. Logical Natural Language Generation from Open-Domain Tables. In *ACL*. 7929–7942.

[11] Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2020. TabFact: A Large-scale Dataset for Table-based Fact Verification. In *ICLR*.

[12] Hyunsoo Cho, Hyuhng Joon Kim, Junyeob Kim, Sang-Woo Lee, Sang goo Lee, Kang Min Yoo, and Taeuk Kim. 2022. Prompt-Augmented Linear Probing: Scaling Beyond The Limit of Few-shot In-Context Learners. In *AAAI*.

[13] Vincent Claveau, Antoine Chaffin, and Ewa Kijak. 2021. Generating artificial texts as substitution or complement of training data. *arXiv preprint arXiv:2110.13016* (2021).

[14] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. (2017). http://archive.ics.uci.edu/ml

[15] Julian Eisenschlos, Syrine Krichene, and Thomas Müller. 2020. Understanding tables with intermediate pre-training. In *EMNLP*. 281–296.

[16] Steven Y Feng, Varun Gangal, Dongyeop Kang, Teruko Mitamura, and Eduard Hovy. 2020. Genaug: Data augmentation for finetuning text generators. *arXiv preprint arXiv:2010.01794* (2020).

[17] Orest Gkini, Theofilos Belmpas, Georgia Koutrika, and Yannis E. Ioannidis. 2021. An In-Depth Benchmarking of Text-to-SQL Systems. In *SIGMOD*. ACM, 632–644.

[18] Vivek Gupta, Riyaz A. Bhat, Atreya Ghosal, Manish Shrivastava, Maneesh Kumar Singh, and Vivek Srikumar. 2022. Is My Model Using The Right Evidence? Systematic Probes for Examining Evidence-Based Tabular Reasoning. *Trans. Assoc. Comput. Linguistics* 10 (2022), 659–679.

[19] Vivek Gupta, Maitrey Mehta, Pegah Nokhiz, and Vivek Srikumar. 2020. INFOTABS: Inference on Tables as Semi-structured Data. In *ACL*. ACL, Online, 2309–2324.

[20] Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 2020. TaPas: Weakly Supervised Table Parsing via Pre-training. In *ACL*. Association for Computational Linguistics, 4320–4333. https://doi.org/10.18653/v1/2020.acl-main.398

[21] Thien Ho Huong and Vinh Truong Hoang. 2020. A data augmentation technique based on text for Vietnamese sentiment analysis. In *International Conference on Advances in Information Technology*. 1–5.

[22] Georgios Karagiannis, Mohammed Saeed, Paolo Papotti, and Immanuel Trummer. 2020. Scrutinizer: A Mixed-Initiative Approach to Large-Scale, Data-Driven Claim Verification. *Proc. VLDB Endow.* 13, 11 (2020), 2508–2521.

[23] George Katsogiannis-Meimarakis and Georgia Koutrika. 2021. A Deep Dive into Deep Learning Approaches for Text-to-SQL Systems. In *SIGMOD*. ACM, 2846–2851.

[24] Karen Kukich. 1983. Design of a Knowledge-Based Report Generator. In *21st Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Cambridge, Massachusetts, USA, 145–150. https://doi.org/10.3115/981311.981340

[25] Varun Kumar, Ashutosh Choudhary, and Eunah Cho. 2020. Data augmentation using pre-trained transformer models. *arXiv preprint arXiv:2003.02245* (2020).

[26] Varun Kumar, Hadrien Glaude, Cyprien de Lichy, and William Campbell. 2019. A closer look at feature space data augmentation for few-shot intent classification. *arXiv preprint arXiv:1910.04176* (2019).

[27] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. (2019). https://doi.org/10.48550/ARXIV.1910.13461

[28] Hao Li, Chee-Yong Chan, and David Maier. 2015. Query from Examples: An Iterative, Data-Driven Approach to Query Construction. *Proc. VLDB Endow.* 8, 13 (sep 2015), 2158–2169. https://doi.org/10.14778/2831360.2831369

[29] Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. 2017. Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision. In *ACL*. 23–33.

[30] Ruibo Liu, Guangxuan Xu, Chenyan Jia, Weicheng Ma, Lili Wang, and Soroush Vosoughi. 2020. Data boost: Text data augmentation through reinforcement learning guided conditional generation. *arXiv preprint arXiv:2012.02952* (2020).

[31] Yinhan Liu, 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* (2019).

[32] Yu Meng, Jiaxin Huang, Yu Zhang, and Jiawei Han. 2022. Generating Training Data with Language Models: Towards Zero-Shot Language Understanding. *CoRR* abs/2202.04538 (2022). arXiv:2202.04538 https://arxiv.org/abs/2202.04538

[33] Yu Meng, Martin Michalski, Jiaxin Huang, Yu Zhang, Tarek F. Abdelzaher, and Jiawei Han. 2022. Tuning Language Models as Training Data Generators for Augmentation-Enhanced Few-Shot Learning. *CoRR* abs/2211.03044 (2022). https://doi.org/10.48550/arXiv.2211.03044 arXiv:2211.03044

[34] Yu Meng, Jiaming Shen, Chao Zhang, and Jiawei Han. 2018. Weakly-Supervised Hierarchical Text Classification. (2018). https://doi.org/10.48550/ARXIV.1812.11270

[35] Preslav Nakov, David P. A. Corney, Maram Hasanain, Firoj Alam, Tamer Elsayed, Alberto Barrón-Cedeño, Paolo Papotti, Shaden Shaar, and Giovanni Da San Martino. 2021. Automated Fact-Checking for Assisting Human Fact-Checkers. In *IJCAI*. ijcai.org, 4551–4558. https://doi.org/10.24963/ijcai.2021/619

[36] Linyong Nan, Lorenzo Jaime Yu Flores, Yilun Zhao, Yixin Liu, Luke Benson, Weijin Zou, and Dragomir Radev. 2022. R2D2: Robust Data-to-Text with Replacement Detection. *arXiv preprint arXiv:2205.12467* (2022).

[37] A. Neveol, Dalianis, and S. Velupillai. 2018. Clinical Natural Language Processing in languages other than English: opportunities and challenges. *Journal Biomed Semantic* 9, 12 (2018).

[38] Liangming Pan, Wenhu Chen, Wenhan Xiong, Min-Yen Kan, and William Yang Wang. 2021. Zero-shot Fact Verification by Claim Generation. In *ACL*. Association for Computational Linguistics, 476–483.

[39] Ankur P. Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. 2020. ToTTo: A Controlled Table-To-Text Generation Dataset. In *EMNLP*. ACL, 1173–1186.

[40] Siyuan Qiu, Binxia Xu, Jie Zhang, Yafang Wang, Xiaoyu Shen, Gerard De Melo, Chong Long, and Xiaolong Li. 2020. Easyaug: An automatic textual data augmentation platform for classification tasks. In *Companion Proceedings of the Web Conference 2020*. 249–252.

[41] Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know What You Don't Know: Unanswerable Questions for SQuAD. In *ACL*. Association for Computational

Linguistics, Melbourne, Australia, 784–789. https://doi.org/10.18653/v1/P18-2124
[42] Ehud Reiter and Robert Dale. 2002. Building Applied Natural Language Generation Systems. *Natural Language Engineering* 3 (03 2002).
[43] Anish Das Sarma, Aditya G. Parameswaran, Hector Garcia-Molina, and Jennifer Widom. 2010. Synthesizing view definitions from data. In *ICDT*. ACM, 89–103.
[44] Dinghan Shen, Mingzhi Zheng, Yelong Shen, Yanru Qu, and Weizhu Chen. 2020. A simple but tough-to-beat data augmentation approach for natural language understanding and generation. *arXiv preprint arXiv:2009.13818* (2020).
[45] Elena Soare, Iain Mackie, and Jeffrey Dalton. 2022. DocuT5: Seq2seq SQL Generation with Table Documentation. *CoRR* abs/2211.06193 (2022). https://doi.org/10.48550/arXiv.2211.06193 arXiv:2211.06193
[46] Wei Chit Tan, Meihui Zhang, Hazem Elmeleegy, and Divesh Srivastava. 2017. Reverse Engineering Aggregation Queries. *Proc. VLDB Endow.* 10, 11 (2017), 1394–1405. https://doi.org/10.14778/3137628.3137648
[47] Immanuel Trummer. 2022. From BERT to GPT-3 Codex: Harnessing the Potential of Very Large Language Models for Data Management. *Proc. VLDB Endow.* 15, 12 (2022), 3770–3773.
[48] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In *ACL*. 7567–7578.
[49] Congcong Wang and David Lillis. 2019. Classification for Crisis-Related Tweets Leveraging Word Embeddings and Data Augmentation.. In *TREC*.
[50] Nancy XR Wang, Diwakar Mahajan, Marina Danilevsky, and Sara Rosenthal. 2021. SemEval-2021 task 9: Fact verification and evidence finding for tabular data in scientific documents (SEM-TAB-FACTS). *arXiv preprint arXiv:2105.13995* (2021).
[51] Nathaniel Weir, Prasetya Utama, Alex Galakatos, Andrew Crotty, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bhushan, Nadja Geisler, Benjamin Hättasch, Steffen Eger, Ugur Çetintemel, and Carsten Binnig. 2020. DBPal: A Fully Pluggable NL2SQL Training Pipeline. In *SIGMOD*. ACM, 2347–2361.
[52] Yaacov Y. Weiss and Sara Cohen. 2017. Reverse Engineering SPJ-Queries from Examples. In *PODS*. ACM, 151–166.
[53] Sam Wiseman, Stuart Shieber, and Alexander Rush. 2017. Challenges in Data-to-Document Generation. In *EMNLP*. ACL, 2253–2263.
[54] You Wu, Pankaj K. Agarwal, Chengkai Li, Jun Yang, and Cong Yu. 2017. Computational Fact Checking through Query Perturbations. *ACM Trans. Database Syst.* 42, 1 (2017), 4:1–4:41.
[55] Qizhe Xie, Zihang Dai, Eduard Hovy, Thang Luong, and Quoc Le. 2020. Unsupervised data augmentation for consistency training. *Advances in Neural Information Processing Systems* 33 (2020), 6256–6268.
[56] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In *ACL*. 8413–8426.
[57] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *EMNLP*. 3911–3921.
[58] Meihui Zhang, Hazem Elmeleegy, Cecilia M. Procopiuc, and Divesh Srivastava. 2013. Reverse Engineering Complex Join Queries. In *SIGMOD*. ACM, 809–820.