A Framework for the Generation of Training Examples from Tabular Data

(Discussion Paper)

Jean-Flavien Bussotti[1], Paolo Papotti[1], Donatello Santoro[2] and Enzo Veltri[2,*]

[1]EURECOM, Biot, France
[2]Università degli Studi della Basilicata (UNIBAS), Potenza, Italy

Abstract

Tabular data is becoming increasingly important in Tabular Natural Language Inference (TNLI), where the goal is to assess if a table supports or refutes a given hypothesis expressed in NL text. A major issue in TNLI is the lack of such training data. Existing approaches are based on manual annotation of new training data or simple augmentation techniques that lack data variety and complexity. We present a system, Tenet, that automatically generates new training examples for TNLI applications on different domains. Our framework exploits SQL queries to introduce new data variety through *evidence-queries* that identify new cell values over data exploiting different data patterns, and complexity using *semantic-queries* that describe the different ways such data can be identified through SQL queries. Description from the semantic-queries are used to verbalize the new cell values from the evidence-queries using a Pretrained Language Model (PLM). The verbalized sentence and the cell values can be used as a new training example in the target TNLI application. We show how Tenet generates human-like examples that are comparable with manually-written examples.

Keywords

Tabular Natural Language Inference (TNLI), Natural Language Processing (NLP) for Databases, Text Generation, Query Generation, Data Augmentation,

1. Introduction

A large class on natural language inference (NLI) problems aims at classifying a given hypothesis, such as a textual statement, as true/false/unknown given some evidence. 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 and computational fact checking, where systems label text claims according to input structured data [1, 2, 3, 4, 5, 6, 7].

Most of the solutions in TNLI are supervised, where manually defined datasets for TNLI have been proposed, such as Feverous [8], TabFact [9], and Infotabs [5]. However, these datasets: i) cover only some generic topics from Wikipedia tables. For example, if there is a need for

SEBD 2024: 32nd Symposium on Advanced Database Systems, June 23-26, 2024, Villasimius, Sardinia, Italy
*Corresponding author.
jflavien.bussotti@gmail.com (J. Bussotti); papotti@eurecom.fr (P. Papotti); donatello.santoro@unibas.it (D. Santoro); enzo.veltri@unibas.it (E. Veltri)
0009-0009-8869-6025 (J. Bussotti); 0000-0003-0651-4128 (P. Papotti); 0000-0002-5651-8584 (D. Santoro); 0000-0001-9947-8909 (E. Veltri)

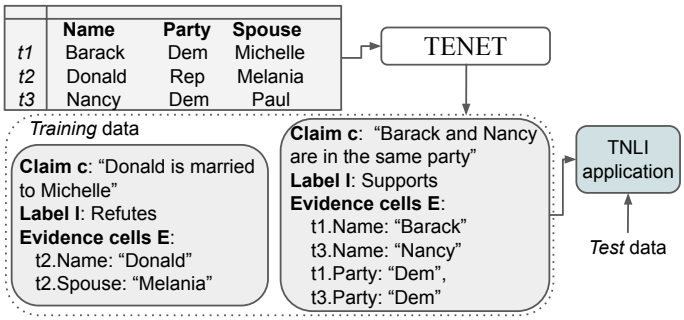


Figure 1: Given any table, TENET generates new training examples for a target TNLI application. The first example has a hypothesis that is refuted according to the data evidence.

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 [10]. For example, about 80% of the examples in Totto [11] 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 [12].

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 [13]. 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 a minor impact on the accuracy of the TNLI tasks. Moreover, in a *cold start* setting, where training data is unavailable, there is no proposal yet on creating annotated examples for TNLI starting only from the tables.

User-provided tables can be exploited to generate ad-hoc training data for the application at hand. Our system, TENET[1] (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 [14]. Figure 1 shows an overview of our architecture. The system generates training data for the target TNLI application, given only a table as input. Once generated, the examples are used to train the inference model 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

[1]Code and datasets available at https://github.com/dbunibas/tenet

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 [15]. Our prompting leads to a variety of factual hypotheses, 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 hypotheses that are refuted w.r.t. the original "clean" data.

In the next Section we describe the main components, then we present some experimental results using TENET.

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[2]. 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 [8] to the entire relation in others [9][3]. 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 [8].

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.

[2]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.

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

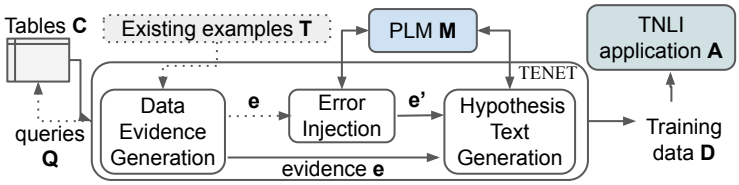


Figure 2: TENET overview. Existing examples are optional. Any text-to-text pre-trained language model (PLM) can be used, e.g., ChatGPT. Any target TNLI application can be supported, e.g., tabular fact-checking.

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 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. A 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. Data evidence with two cell values, e.g., Name for tuple t_1 and Age from tuple t_2 can lead to a 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. We call this approach *cold-start*. One approach to improve this task and obtain meaningful patterns with fewer generated examples is to infer data patterns from human-provided examples T, when those are available. For example, in T, we identify a query q (named *evidence query* or simply *e-query*) that returns the cell values in its data evidence as one result row. We then execute the e-query over the relation. The e-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. We call this approach *warm-start*.

Warm Start. While the cold-start is easy to implement, the generation of the e-query in the warm start is not trivial. 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 evidence $e_1, \ldots, e_n$ 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
```

The e-query generation is based on an evidence graph [14] where each node in the graph corresponds to a cell in the evidence e_s and a (direct) edge across two nodes represents the relationship between their values (equality, difference, comparison). Then visiting such graph we construct the e-query [14].

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 [11], such as "Mike is 47 years old and Anne 22.". While these kinds of sentences are fundamental, we aim to maximize the variety in the training data. For this goal, we generate various queries that return evidence e given c. We call such queries *semantic queries* or simply *s-queries*. Such s-queries represent different ways of semantically describing the data. PLMs are trained over huge amounts of textual data, which gives them proficiency in writing, and source code, which gives them the ability to write code [16] or to be instructed with functions. We then propose prompting methods for PLMs to generate alternative sentences to describe the evidence set according to the semantics of the queries.

We identify several types of s-queries: 1) the surface s-queries, i.e. queries that select cells by using only constant values; 2) comparison s-queries, i.e. queries that compare two or more rows by at least one attribute; 3) filter s-queries, i.e., queries that select cells according to a condition; 4) aggregate s-queries, i.e., queries that select cells that can be used with an aggregative function (count, sum, avg, min, max); 5) filter-aggregate s-queries, i.e., queries that select cells for an aggregation over a group of cells identified by a selection on some conditions. Such s-queries are automatically detected by TENET [14].

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 task uses the function from the s-query and the evidence. The text generation functions mapped to the relative s-queries are reported in Table 2 with examples of the text they generate. Due to space limits, the examples of the used prompts with ChatGPT are reported in the full paper [14].

Table 2
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.

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. The methods above produce Support examples. 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. 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' using the same e-query used for the Support example. A hypothesis h' is then derived from e' using the same proposed approach above. 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 the label Refutes, c, h' and evidence e.

To inject errors, 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 involved in e. This step breaks the original relationships across cell values at the tuple level. We then either introduce a new tuple in c' 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. 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. Finally, we remove from c' any row that appears in c.

3. Experiments and Conclusions

We organize our evaluation around two main questions. First, does TENET automatically generate training data of quality comparable to those manually created by human annotators? Second, what are the costs of TENET, in terms of execution time and budget for external APIs?

Train Datasets. In this paper we present results for a dataset from TNLI literature: FEVEROUS [8]. Results for other datasets are presented in the full paper [14]. FEVEROUS 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* (a subset of the table), and a Supports/Refutes *label*. All examples are manually written by humans.

As a baseline, we extend the original training dataset with an augmentation for text [17]. 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 dataset for our techniques. Given a corpus of tables, we always generate the *Tenet Cold* (*TenetC*) dataset. Since FEVEROUS has annotations for data evidence, we can also generate the dataset for *Tenet Warm* (*TenetW*). Hypotheses are created with s-queries and negative examples are generated according the presented technique. For each given table, we produce three Supports and three Refutes hypotheses, therefore all TENET datasets are balanced in terms of labels. For every table, TENET creates one example with a surface query and two with other s-queries among the other four types (Comparison, Filter, Aggregate, FilterAggregate).

Inference Models for TNLI. 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. For FEVEROUS the inference predictor is a RoBERTa (large) encoder fine-tuned for classification on multiple NLI datasets [18].

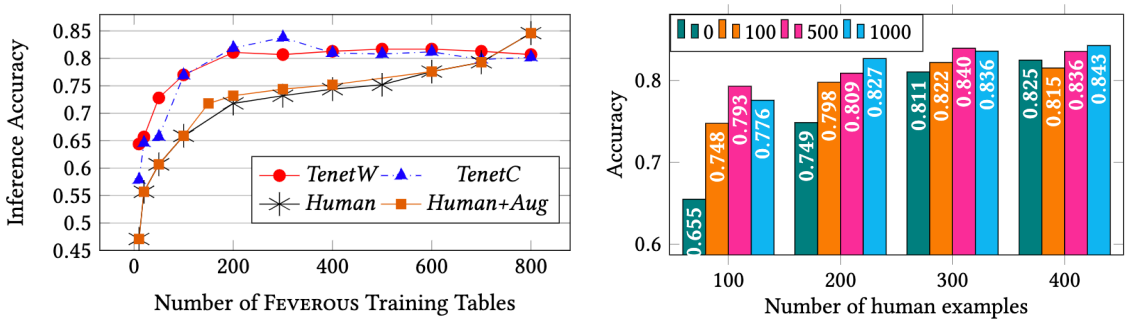
Pre-trained Language Models. For the hypothesis generation and the error injection, we assume that a pre-trained language model (PLM) is available. We tested several PLMs and use ChatGPT as default. We report a comparison of T5, fine-tuned on ToTTo, and ChatGPT in the full paper [14].

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.

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 dataset. 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 on the same test data.

The TNLI accuracy results in Figure 3a 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.

Figure 3b 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.



(a) (b)

Figure 3: (a) 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. (b) 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.

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.

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. On average the Cold takes 2.019 seconds while Warm takes 2.212. 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 3
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 3 reports the costs of generating hypotheses 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 the full paper [14] is about $130 for 36K generated examples.

Conclusions. We presented 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.

References

[1] G. Karagiannis, M. Saeed, P. Papotti, I. Trummer, Scrutinizer: A mixed-initiative approach to large-scale, data-driven claim verification, Proc. VLDB Endow. 13 (2020) 2508–2521.

[2] Y. Wu, P. K. Agarwal, C. Li, J. Yang, C. Yu, Computational fact checking through query perturbations, ACM Trans. Database Syst. 42 (2017) 4:1–4:41.

[3] P. Nakov, D. P. A. Corney, M. Hasanain, F. Alam, T. Elsayed, A. Barrón-Cedeño, P. Papotti, S. Shaar, G. D. S. Martino, Automated fact-checking for assisting human fact-checkers, in: IJCAI, ijcai.org, 2021, pp. 4551–4558. URL: https://doi.org/10.24963/ijcai.2021/619. doi:10.24963/ijcai.2021/619.

[4] J. Herzig, P. K. Nowak, T. Müller, F. Piccinno, J. Eisenschlos, TaPas: Weakly supervised table parsing via pre-training, in: ACL, Association for Computational Linguistics, 2020, pp. 4320–4333. URL: https://aclanthology.org/2020.acl-main.398. doi:10.18653/v1/2020.acl-main.398.

[5] V. Gupta, M. Mehta, P. Nokhiz, V. Srikumar, INFOTABS: Inference on tables as semi-structured data, in: ACL, ACL, Online, 2020, pp. 2309–2324.

[6] E. Veltri, D. Santoro, G. Badaro, M. Saeed, P. Papotti, Pythia: Unsupervised generation of ambiguous textual claims from relational data, 2022, p. 2409 – 2412. doi:10.1145/3514221.3520164.

[7] E. Veltri, G. Badaro, M. Saeed, P. Papotti, Data ambiguity profiling for the generation of training examples, in: 39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, CA, USA, April 3-7, 2023, IEEE, 2023, pp. 450–463. URL: https://doi.org/10.1109/ICDE55515.2023.00041. doi:10.1109/ICDE55515.2023.00041.

[8] R. Aly, Z. Guo, M. S. Schlichtkrull, J. Thorne, A. Vlachos, C. Christodoulopoulos, O. Cocarascu, A. Mittal, FEVEROUS: Fact extraction and VERification over unstructured and structured information, in: NeurIPS (Datasets and Benchmarks), 2021.

[9] W. Chen, H. Wang, J. Chen, Y. Zhang, H. Wang, S. Li, X. Zhou, W. Y. Wang, Tabfact: A large-scale dataset for table-based fact verification, in: ICLR, 2020.

[10] P. Rajpurkar, R. Jia, P. Liang, Know what you don't know: Unanswerable questions for SQuAD, in: ACL, Association for Computational Linguistics, Melbourne, Australia, 2018, pp. 784–789. URL: https://aclanthology.org/P18-2124. doi:10.18653/v1/P18-2124.

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

[12] V. Gupta, R. A. Bhat, A. Ghosal, M. Shrivastava, M. K. Singh, V. Srikumar, Is my model using the right evidence? systematic probes for examining evidence-based tabular reasoning, Trans. Assoc. Comput. Linguistics 10 (2022) 659–679.

[13] M. Bayer, M.-A. Kaufhold, C. Reuter, A survey on data augmentation for text classification, ACM Computing Surveys (2022).

[14] J.-F. Bussotti, E. Veltri, D. Santoro, P. Papotti, Generation of training examples for tabular natural language inference, Proc. ACM Manag. Data 1 (2023). URL: https://doi.org/10.1145/3626730. doi:10.1145/3626730.

[15] I. Trummer, From BERT to GPT-3 codex: Harnessing the potential of very large language models for data management, Proc. VLDB Endow. 15 (2022) 3770–3773.

[16] G. Mecca, D. Santoro, N. Sileno, E. Veltri, Diogene-ct: tools and methodologies for teaching

and learning coding, International Journal of Educational Technology in Higher Education 18 (2021). doi:10.1186/s41239-021-00246-1.

[17] J. Eisenschlos, S. Krichene, T. Müller, Understanding tables with intermediate pre-training, in: EMNLP, 2020, pp. 281–296.

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