Fact-Checking Statistical Claims with Tables

Mohammed Saeed and Paolo Papotti
EURECOM, France

Abstract

The surge of misinformation poses a serious problem for fact-checkers. Several initiatives for manual fact-checking have stepped up to combat this ordeal. However, computational methods are needed to make the verification faster and keep up with the increasing abundance of false information. Machine Learning (ML) approaches have been proposed as a tool to ease the work of manual fact-checking. Specifically, the act of checking textual claims by using relational datasets has recently gained a lot of traction. However, despite the abundance of proposed solutions, there has not been any formal definition of the problem, nor a comparison across the different assumptions and results. In this work, we make a first attempt at solving these ambiguities. First, we formalize the problem by providing a general definition that is applicable to all systems and that is agnostic to their assumptions. Second, we define general dimensions to characterize different prominent systems in terms of assumptions and features. Finally, we report experimental results over three scenarios with corpora of real-world textual claims.

1 Introduction

Large scale spreading of incorrect information on the internet is a real threat that poses severe societal problems [30]. As no barriers exist for publishing information, it is possible for anyone to diffuse false or biased claims and reach large audiences with ease [6]. This raises the important issue of how to tame the spread of false information, as this has affected public votes[1] and has misinformed people about coronavirus remedies[2] and spread[3]. Accordingly, there has been a great demand for fact-checkers to efficiently verify such news.

Indeed, with the easy accessibility of large social networks and the advent of generating text using recent advances in Natural Language Processing (NLP) [12,32], the surge of false news has overpowered the capabilities of manual fact-checking. Malicious users in social networks are still allowed to profit from misinformation and the affected networks have just started to take effective actions [1]. At the beginning of the COVID-19 pandemic, the spread of false coronavirus news has urged the World Health Organization to spotlight this issue, labeling it as an *infodemic* [2]. One approach to deal with this enormous volume of information is computational fact-checking [34], where parts of or the entire verification pipeline is automated, usually including some ML algorithms [25]. One influential system is ClaimBuster [14], which is an end-to-end fact-checking solution that relies on NLP and supervised algorithms to identify and check factual and false information. Since then, there has been a stream of fact-checking systems.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[1] https://www.ucf.edu/news/how-fake-news-affects-u-s-elections/
[2] https://fullfact.org/health/honey-ginger-pepper-WHO/
[3] https://fullfact.org/health/indian-variants-sequencing/

Player	Minutes Played	Field Goals	Field Goal Attempts	Assists	Points$^{(3)}$
Courtney Lee	39:08	9	14	3	22
Marc Gasol	35:23	$6^{(2)}$	$12^{(2)}$	6	18
Zach Randolph	29:26	4	9	0	10
Mike Conley	29:13	9	14	$11^{(1)}$	$24^{(4)}$
Tony Allen	23:10	4	6	1	9
Quincy Pondexter	26:43	2	8	0	7
Beno Udrih	18:47	3	6	3	6
Jon Leuer	16:13	1	4	0	2
Kosta Koufos	12:37	0	2	1	0
Vince Carter	9:20	2	5	0	4

Table 4: Statistics of a basketball game.

Some systems verify a certain input by utilizing structured data stored in *knowledge graphs* [5,16] or *relational tables* (or just *relations*) [18, 19], while others rely on unstructured resources such as Wikipedia articles [27, 36]. Other systems have the ability to verify multiple claims occurring within the same input text, such as an entire document [15, 19]. A recent approach also discards all evidence retrieval methods and relies on the implicit knowledge stored in large pre-trained language models (PLM) [20]. Nevertheless, the plethora of different methods calls for a thorough study of their differences with an exploration of the salient aspects related to the design of the systems and how they relate to the fact-checking process. Such study aims to (i) provide readers with a set of dimensions to model this kind of systems, (ii) inspect prominent systems by testing them on various datasets.

We are interested in claims that can be verified by using existing relational tables. Storing data in tables is the go-to format for many applications as it offers declarative querying, scalability, and several other features. For example, reliable statistics for coronavirus are published on a daily basis as relations [4]. The use of such tables supports the verification process and can help in relieving the work done by human fact-checkers. Indeed, manually verifying textual reports, which summarize the most important statistics, is time-consuming and requires automation [3, 25].

In this article, we start with formulating the problem of the computational verification of textual claims by using relational data (Section 2). We discuss four recent systems and highlight their main differences in terms of six generic dimensions (Section 3). We then experimentally compare the systems on several annotated claim corpora (Section 4). Finally, we conclude with some open challenges and future directions for research in this topic (Section 5).

2 Problem Statement

We introduce our problem and related terminology. We assume a scenario where we have a natural language text containing one or more claims to be verified with some relational table(s). Such table(s) are either given as an input or predicted by a system. The following example contains (hypothetical) claims about a basketball game. Table 4[5] contains the information needed to verify such claims. Table values that are used to verify a claim are marked with the same superscript.

Example 1: "Mike Conley had $11^{(1)}$ assists. The field goal percentage for Marc Gasol is $50\%^{(2)}$. The team scored $100^{(3)}$ points in total. Mike Conely scored the $most^{(4)}$ points."

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

[5] Obtained from `https://www.basketball-reference.com/boxscores/201411050PHO.html`

The first claim can be verified with a simple look-up in the table over the **Assists** attribute. The second claim can be verified by computing the ratio of the **Field Goals** to the **Field Goal Attempts** of a certain player; thus, two cell values are needed for verification. The third (false) claim can be checked with an aggregation (summation) over the **Points** column. The fourth claim involves finding the player with the maximum number of **Points**. Other claims might need the involvement of two or more tables.

Definition 1: *Fact-checking* is the process of checking that all facts in a piece of text are correct.

Fact-checking is usually composed of three phases: (i) finding check-worthy claims, (ii) finding the best available evidence for the claim at hand, and (iii) validating or correcting the record by evaluating the claim in light of the evidence [25]. Claims can be provided in a structured form, such as the subject-predicate-object triples in a knowledge graph [5], or in plain text [31], such as the sentences in Example 1. In this article, we assume that the first step (i) has already been executed, and every sentence contains at least one claim.

Definition 2: A *textual claim* is any subset of a natural language input that is to be verified against trustworthy reference sources.

Data in such reference sources can be structured or non-structured. Non-structured data include textual documents while structured data include knowledge graphs, such as DBpedia [21], and relational tables. In this work, we are interested in relational tables as reference data. Specifically, we focus on tables that contain numerical data and on which numerical and Boolean operations can be computed.

Definition 3: A *statistical claim* is any textual claim that can be verified over a trustworthy database by performing a mathematical expression on the database cell values.

The claims in Example 1 are all statistical claims, while a claim such as “Players who commit too many fouls are not allowed to play anymore." is not.

Definition 4: An *explicit claim* is a statistical claim mentioning a number that can be verified by comparing it against the result of a function that takes as parameters some cell values in the input relation.

The first three claims in Example 1 are explicit claims. We assume that symbols `LOOKUP`, `SUM`, and `DIVISION` are defined. `LOOKUP` performs a look-up in a table given a primary key value and an attribute label. `SUM` performs a summation over the values of an attribute. `DIVISION` performs the division of two cell values. The first claim is a simple look-up over the table that could be modeled as `LOOKUP('Mike Conley','Assists')==11`. The second claim requires computing a ratio of Field Goals to Field Goal Attempts for player Marc Gasol. This can be formulated as `DIVISION(a,b)==0.5` where `a=LOOKUP('Marc Gasol','Field Goals')` and `b=LOOKUP('Marc Gasol','Field Goal Attempts')`. The third claim could be modeled as `SUM('Points')==100`.

Definition 5: An *implicit claim* is a statistical claim that does not mention a number and can be verified by a Boolean function that takes as parameters cell values in the input relation.

The last claim in Example 1 is an implicit claim. Assuming MAX has also been defined, it could be modeled as `MAX(Points)=LOOKUP('Mike Conely','Points')`. As we will discuss in the next section, implicit claims are harder to verify and usually require some form of supervised learning, such as the learning of neural representations [19] or the synthesis of a program from the input [8].

Definition 6: Given a text T containing a statistical claim c and a database D, the goal of *Statistical-Claim Fact-Checking* is to verify c with the information in D. Formally, the objective is to find a function $f(T, c, D)$ that successfully maps to one of two labels (**True** or **False**).

This definition is generic enough to model existing fact-checking systems and other systems that can be adapted for this task[6]. It is independent of the different assumptions that apply for the different approaches. For example, multiple systems assume that the input text T contains a single claim [8], thus dropping the need for having the claim c as an input. Also, the database D is often simplified to one relational table given as input [8, 15], while other systems utilize multiple tables [18, 19]. Finally, the definitions above do not cover a notion of explainability of the provided result. Indeed, some systems do not provide any result explanation since the verification process relies on black-box methods, such as deep neural networks [8].

3 Systems

In this section, we study four systems that satisfy Definition 6. We analyze TABLE-BERT [8], TAPAS [15], AGGCHECKER [18], and SCRUTINIZER [19]. In Section 3.1, we describe how each system works and its assumptions. In Section 3.2, we introduce six dimensions to characterize such fact-checking systems.

3.1 Overview of Systems

We introduce the systems, starting from the ones that rely on end-to-end NLP methods, and describe those that use query generation next.

TABLE-BERT [8] models fact-checking of a statistical claim as a Natural Language Inference (NLI) problem [22]. NLI is the task of determining whether a natural language hypothesis h can be inferred from a natural language premise p. In TABLE-BERT, a given table **T** is linearized and fed to the model alongside the natural language hypothesis p. The model consists of a pre-trained BERT model [12] that outputs a sequence-level representation of the input. This representation is then fed into a multi-layer perceptron, which predicts the entailment probability. If the output probability is greater than 0.5, then the hypothesis p is entailed by table **T**.

This system assumes a table as input, i.e., that the reference data is available and already identified. TABLE-BERT comes with a corpus of tables and claims (annotated as true/false) that can be used for fine-tuning. This makes it usable on unseen tables, but our experimental results show that further fine-tuning for the domain at hand is needed to obtain good results. Moreover, TABLE-BERT can be fine-tuned with more examples that contain formulas unseen in the provided corpus. However, the original paper recognizes that the complexity of the formulas that the system can learn is limited and does not support composition of functions [8].

TAPAS [15] can be used to tackle the claim verification as a question-answering problem over an input table [4, 23, 33]. The model takes as input (1) a natural language question **Q** to be answered over (2) the input table **T**. Building on the success of pre-training models on textual data, TAPAS extends this procedure to structured data, by training a BERT [12] model on a large number of natural language questions and Wikipedia tables. This process enables the model to learn correlations between structured and unstructured data. After training, the encoder provides a representation of the input. The output is twofold: the model predicts (1) which *cell values* of the input table are used for answering the question and (2) what *aggregation operation* is performed on such values to produce an answer for the input question.

As with TABLE-BERT [8], TAPAS assumes that the reference table is available, and is linearized in the input. Generating questions from the claim could be done using lexical-based methods as pioneered in ClaimBuster [13], or neural-based methods [7, 35]. While TAPAS has the benefit of being general, i.e., “plug and play" on new domains, it has the limitation that extending it to new formulas or tables requires the full re-training from scratch. Moreover, it is not demonstrated that it could learn complex formulas.

[6]While the definition considers a binary label for the output, it can be extended to multiple labels such as “partially true” or “not enough evidence".

AGGCHECKER [18] takes a relational dataset and a text document as input. It translates the natural language claims in the document into SQL queries that are executed to verify the claims. More specifically, each claim is mapped to a probability distribution over SQL queries. SQL queries are formed by combining query fragments using an iterative expectation-maximization procedure [11]. AGGCHECKER works out of the box on unseen relations and does not assume that training data is available for a new database. The system supports aggregation functions and could be extended to support more. Extending the system is not trivially done by just feeding more binary examples. It needs an update to the information retrieval engine to incorporate new query fragments, and an update to the probabilistic model to account for new SQL query candidates. A module to account for multi-variable formulas is also needed. Similar to TAPAS, the modification needed for this system to account for unseen functions goes beyond examples.

The system benefits from the fact that claims in the same context are often semantically correlated by learning a document-specific prior distribution over queries. As in practice accurate claims are more likely than inaccurate claims, the system increases the likelihood of the query which has a match between the query result and the claim. As multiple candidate queries are to be executed, an execution engine that merges execution of similar queries is used for efficiency.

SCRUTINIZER defines fact-checking as a mixed-initiative verification problem [19]. The approach combines feedback from human workers with automated verification coming from ML classifiers. We neglect the human-in-the-loop part in this article and focus solely on the automatic verification. The system is based on four classifiers that take a statistical claim as input and predict (i) the relation(s) to be used, (ii) the primary key value(s), (iii) the attribute label(s), and (iv) the formula applied on the cells identified by the former three. In contract with other systems, the table is not given as input, but is predicted, and the cell selection is based on the predicted primary key values and attribute labels for such a table. This leaves out the need for inputting the table to the model, but limits the current system only to the table schemas seen during training. After cell selection is done, it can apply the predicted formula and verify the input claim.

SCRUTINIZER can learn any query, including complex formulas, from the training data. However, the price to pay for this generality is that it trains the classifiers, therefore labels for these must be provided, and it does not suffice to have the true/false label for the claim as in TABLE-BERT.

Aside from AGGCHECKER, all the systems use transformer-based language models [12] to encode language knowledge, but only TAPAS requires the expensive pre-training of such models. AGGCHECKER relies on a probabilistic model to map natural language claims to a probability distribution over queries. Others solutions rely on synthesizing a logical program [8], recurrent-based language models [26], reinforcement-learning approaches [37], and graph neural-networks [24].

Type	Dimension	AGGCHECKER	TAPAS	TABLE-BERT	SCRUTINIZER
Input	Implicit Claims		✓	✓	✓
	Schema-Independence	✓	✓	✓̸	
	Multi-variable Formulas	✓̸	✓̸		✓
	Multi-tables	✓̸			✓
Output	Interpretability	✓	✓̸		✓
	Alternative Interpretations	✓			✓

Table 5: Dimensions that characterize the systems (✓̸ denotes partial support).

3.2 System Dimensions

We believe that, given the increasing number of fact-checking systems, it is important to start characterizing them with clear dimensions to enable a more rigorous comparison. We first describe four main dimensions that

characterize the input across the different proposals. Then, we discuss two dimensions that characterize the output. A summary of the dimensions and how systems support them is reported in Table 5.

3.2.1 Input Dimensions

Explicit claims are handled by all fact-checking systems, as they are much easier to deal with. However, the support for **Implicit Claims** requires a deeper understanding of the semantics behind the given sentence. One approach to dealing with this problem is feature-based entity linking where all entities are detected in the input statement and a set of pre-defined trigger-words are used to build programs representing the semantics of the statement [8]. However, such approaches are very sensitive to the error-prone entity linking process. Another approach is to learn such implicit claims in a supervised manner. SCRUTINIZER learns from the classifiers' labels [19]. TAPAS also learns correlations between the text and the table during the pre-training process.

Another dimension is **Schema-Independence**. AGGCHECKER, TABLE-BERT and TAPAS can consume potentially any table with any unseen schema, while SCRUTINIZER is limited to tables whose row index values and attribute labels have been trained on. For SCRUTINIZER, adding new tables requires fine-tuning the classifiers. The operation is not expensive in terms of execution time, because its classifiers are based on a fine-tuning procedure, rather than having to pre-train again from scratch; however, it requires specific annotations that go beyond the true/false binary label. This dimension highlights that SCRUTINIZER is domain-specific and thus has to learn the related tables for the task at hand, while AGGCHECKER and TAPAS try to be agnostic of the table schema, and can handle any table as input. For TABLE-BERT, while it can be used on any unseen schema, our experiments show that it should be trained on the examples at hand in order to obtain good accuracy performance.

In practice, computations involving values of a database go beyond simple look up and aggregation functions such as those reported in Example 1. The function for the verification of a claim can require complex **Multi-variable Formulas**. For example, the Compound Annual Growth Rate[7] is a formula needed to verify a claim in our experiments. SCRUTINIZER handles complex formulas on the condition that they are observed in its classifier-specific training data, and resorts to a brute-force approach to assign predicted values to variables. AGGCHECKER can be extended to handle complex aggregation functions. TAPAS handles aggregate queries with simple functions where the cell values have been selected by the model. It is not clear if and how TAPAS could support functions with more than one variable, and it would require training again the model from scratch such that new functions are learned. Finally, TABLE-BERT has no explicit notion of formulas, as it is a black-box model fine-tuned end-to-end on a binary classification task. According to the original paper and our experiments, TABLE-BERT struggles to learn how to handle formulas with multiple variables.

TABLE-BERT and TAPAS assume that the right table to verify the input claim is also given as input. In practice, many tables can be available and the most likely one for the task at hand is identified by SCRUTINIZER and AGGCHECKER (**Multi-Tables**). Moreover, in some cases more than one table is needed to verify a claim and only SCRUTINIZER supports verification that requires the combination of values from multiple tables. This dimension highlights one of the limits of the methods that rely on the linearized data fed to the transformers, as it is hard to feed multiple tables without hitting the limit on the size of the input.

3.2.2 Output Dimensions

Interpretability is a key dimension supported by methods that output the query used to verify the claim. However, systems using a black-box model to verify claims, such as TABLE-BERT, lack interpretability as an explanation of the prediction is not provided. There do exist methods attempting to explain black-box models which include explanations by simplification [29]. However, there is no consistent method to define how faithful are the

[7] It describes the net gain or loss of an investment over a certain period of time (`https://en.wikipedia.org/wiki/Compound_annual_growth_rate`).

explanations to the model prediction [17]. TAPAS is not fully interpretable since it provides only cell values and, in some cases, the aggregation operation. AGGCHECKER and SCRUTINIZER expose the declarative query used to verify the associated claim. Systems that predict query fragments and combine them, rather than producing an answer in one shot, are easier to interpret [10].

Claims expressed in natural language can be incomplete or ambiguous in many ways. Some systems support **Alternative Interpretations** to clarify how the output changes depending on the details of the verification. Consider a simple claim "Mike scored 30 points", and a table with two players whose first name is "Mike". The claim is true for one player, but false for the other. AGGCHECKER resolves such ambiguities by evaluating multiple queries and soliciting feedback from users. SCRUTINIZER learns ambiguities conditioned that they are represented in the training data. TAPAS and TABLE-BERT do not include any clear means to resolve this kind of ambiguities, as they default to one interpretation in the current architectures.

4 Experimental Evaluation

We evaluated the four systems above by using three datasets with real textual claims manually annotated with the correct checking label. We concatenate the claim to the sentence in case the sentence has multiple claims; otherwise, we only input the sentence. For every system, we report its coverage of the claims, the accuracy of the verification process, and the execution times.

		SCRUTINIZER Labels				TABLE-BERT Label
Sentence	Claim	Table	Attribute Label	Primary Key Value	Formula	Verdict
There were 800 total deaths in China in May 2021.	800 total deaths	total_deaths	May_2021	China	a	False

Table 6: Labeled data for SCRUTINIZER and TABLE-BERT.

4.1 Datasets

Our experiments are based on three use cases: Coronavirus scenario (C19), International Energy Agency (IEA) scenario, and Basketball Data scenario (BBL). Every example contains the textual claim, the relational table to verify it, and the outcome expressed as a binary label True/False. For SCRUTINIZER, the training examples contain also the labels for the four classifiers. For a fair comparison, we fed the associated relation as input to all systems. Given the limitations on the input size in TAPAS, we limit the input for this system to at most a sample of 11 tuples, including the one needed to verify the claim. Without this ad-hoc operation, the system fails with the entire relation as input. An example of our labeled data is shown in Table 6.

For C19, we generated the training data from the relations (3M examples [19]) and used real claims for the testing. We denote the synthetic corpus as $\mathbf{C19_{train}}$. For testing the system with unseen claims, we analyzed the log of more than 30K claims tested by users on a website[8]. We found that more than 60% of the claims in such corpus are statistical and, among those, we have the datasets to verify 70%. From these claims, we randomly selected 55 claims and manually annotated them ($\mathbf{C19_{test}}$).

For IEA, we obtained a document of 661 pages, containing 7901 sentences, and the corresponding corpus of manually checked claims, with check annotations for every claim from three domain experts. The annotations cover 2053 numerical claims, out of which we identified 1539 having a formula that occurs at least five times in the corpus. We denote the resulting dataset as $\mathbf{IEA_{train}}$. After processing the claims, we identify 1791 relations, 830 row indexes, 87 columns, and 413 formulas. Around 50% of the values for all properties appear at most 10

[8] `https://coronacheck.eurecom.fr`

times in the corpus, with the top 5% most frequent formulas appearing at least 8 times. For the test data ($\mathbf{IEA_{test}}$), we randomly selected 20 claims from the most common operations (look up and sum).

For BBL, we use the data in a recent publication [28][9]. We use the 1523 real annotated claims provided in the repository for the testing step and generate ourselves the training data from the tables as in the C19 scenario. We generate an initial dataset of 32.3K samples, where 90% is used for training classifiers and 10% for validation. We use 132 tables for this scenario. The dataset used for bootstrapping is denoted by $\mathbf{BBL_{train}}$ and the test dataset as $\mathbf{BBL_{test}}$. Our datasets ($\mathbf{BBL_{test}}$ and $\mathbf{C19_{test}}$) are publicly available[10].

Table 7: Ratio of supported training claims.

	AGGCHECKER	TAPAS	TABLE-BERT	SCRUTINIZER
$\mathbf{C19_{train}}$	21.49%	21.49%	37.82%	100%
$\mathbf{IEA_{train}}$	33.06%	17.02%	27.28%	74.96%
$\mathbf{BBL_{train}}$	53.00 %	56.17%	56.17%	56.17%

As discussed in Section 5, the tested systems have limitations on the input data and on the space of supported formulas. These limitations are reflected in the percentage of training claims that every system can handle, as reported in Table 7. For example, complex formulas are present in our datasets, with more than 22% of the claims in $\mathbf{IEA_{train}}$ with three or more variables.

4.2 Experimental Results

For TABLE-BERT, we fine-tuned the binary classifier on top of the PLM with the training data after augmenting the data to ensure a balance between classes. For TAPAS, we tried to automatically translate the claims to questions as pioneered by ClaimBuster [13]. However, the precision was not satisfactory, e.g., we could not obtain any questions for 7 out of the 20 IEA test claims. To overcome this issue, we manually translated claims into questions for $\mathbf{IEA_{test}}$ and $\mathbf{C19_{test}}$, and relied on a pattern-based script to generate questions for $\mathbf{BBL_{test}}$. For TAPAS and AGGCHECKER, we did not run any training. For SCRUTINIZER, we provided the examples with the labels for the 4 classifiers, and examples with binary labels for TABLE-BERT. For SCRUTINIZER, we do not rely on the user feedback in this experiment.

Table 8: Verification accuracy on the test datasets.

	AGGCHECKER	TAPAS	TABLE-BERT	SCRUTINIZER
$\mathbf{C19_{test}}$	0.44	0.64	0.76	0.80
$\mathbf{IEA_{test}}$	0.50	0.07	0.58	0.65
$\mathbf{BBL_{test}}$	0.13	0.41	0.17	0.51

Table 8 reports the accuracy results of the experiments with all systems on the test claims. $\mathbf{IEA_{train}}$ is heavily skewed as, for instance, there are formulas such as lookups and summations that are commonly used, unlike formulas comprising multiple variables which are scarce (formulas having ten or more variables form 4.32% of the training data). The formulas in $\mathbf{IEA_{test}}$ are different, as they contain functions supported by all systems

[9] `https://github.com/ehudreiter/accuracySharedTask`; sentences and claims in this corpus are similar to the ones reported in Example 1.

[10] `https://zenodo.org/record/5128604#.YPrSgXUzZuU`

Table 9: Execution time of the test datasets (seconds).

	AGGCHECKER	TAPAS	TABLE-BERT	SCRUTINIZER
C19$_{\text{test}}$	280.41	991.52	23.09	0.03
IEA$_{\text{test}}$	321.53	943.25	18.11	0.68
BBL$_{\text{test}}$	3472.44	12709.45	40.20	236.64

(lookups and summations in this case). For **BBL$_{\text{test}}$**, low results are explained by the fact that all systems have low coverage of the claims in the data and some claims require verification that spans across multiple tables. For **C19$_{\text{test}}$**, the systems do slightly better as the test data comprises lookups and the attribute label and primary key value are usually explicitly stated in the sentence. We observe that the systems perform with mixed results, and none of them can get high accuracy in all cases. We can observe that in most cases the use of training data can lead to the best performance. This is evident for TABLE-BERT, which performs well in two datasets by making use of the true/false labels, and in SCRUTINIZER, which exploits the rich annotations for its classifiers and leads in all scenarios. However, for **C19$_{\text{test}}$**, SCRUTINIZER fails for claims which require formulas that it has not seen in the training data. For running **BBL$_{\text{test}}$** with TAPAS and TABLE-BERT, we replaced abbreviations in the schemas of the table by their proper wording (for example, “PTS" was replaced by “Points"). This has improved the TAPAS accuracy from 0.19 to 0.41 and TABLE-BERT accuracy from 0.09 to 0.17. This is expected as such models, which have been trained on text and table together, correlate better a table schema containing “Points" with the input text compared to the acronym “PTS".

For the **execution times**, we distinguish the training and the testing. Classifier-training time is needed for SCRUTINIZER and TABLE-BERT; however, this is typically negligible (on the studied datasets) with the usage of GPUs. AGGCHECKER and TABLE-BERT, on the other hand, have zero setup time. We report the execution times for all test data in Table 9. TAPAS is the slowest as the model is jointly computing the relevant cells and performing an operation on them, compared to TABLE-BERT that requires a negligible amount of time to perform binary classification. SCRUTINIZER consumes negligible time in classifier predictions, but the brute-force query generation process could potentially take considerable amount of time when multiple combinations are available. AGGCHECKER, although having to perform evaluations of a large number of queries, successfully merges the execution of similar queries to increase efficiency. In summary, all systems are usable in reasonable time in our experience in an entry-level infrastructure with a low-end GPU.

5 Conclusions

We focused our study on the problem of fact-checking a statistical claim with relational tables as reference data and considered four prominent systems. We make a first step towards categorizing fact-checking systems with generic dimensions. We have also experimentally evaluated the four systems on three use cases and gathered many observations on their coverage, their qualitative performance and their execution times.

Our results and the proposed categorization can act as a blueprint when designing a system, as different applications have different requirements. For example, text coming from the basketball data scenario is unlikely ambiguous, so it is safe to neglect ambiguity resolution. However, text related to coronavirus is oftentimes ambiguous and resolution of the ambiguities is a must. Data-driven approaches excel with the provision that sufficient training data is accessible; however, this condition is not always easily met, as manual annotation is costly, especially in scenarios such as IEA where experienced labor is needed. Experiments highlight that training data generated from the tables is a valid solution, but it requires manual work. This aspect is especially important for SCRUTINIZER, which has the highest accuracy, but it is the system that requires most labeling efforts. We also

remark that some systems worked only after pre-processing the input, by rewriting the claim as a question or by limiting and refining the tabular data. We can state that there is no one system that clearly fits for all scenarios. Choosing or designing a system has to be done keeping in mind the scenario(s) at hand.

Finally, we discuss a promising research direction that we identified during the experimental campaign. Given that the systems are getting better at detecting a false claim, is there any hope that they learn how to *repair* a false claim with the correct information?

Consider a basketball data scenario with claim "Vince Carter scored 22 points in 39 minutes." Having a look at Table 4, we see that the player scored 4 points in 9 minutes. A fact-checking system would mark the claims as false. However, we see in the table that another player (Courtney Lee) is the actual fit for the sentence. The sentence is still false, but if we aim at repairing it, the result will be very different. In one case we would change the values and in one case we would repair the claim with a different entity — which one is the correct fix? Looking at this example, someone may argue that the mistake is in the entity, following the principle that it is more likely to have one error rather than two in the same sentence. This is in line with several data cleaning systems for relational data, which repair tuples according to a minimality principle [9]. This aspect of fact-checking is not considered in the design of the current systems, nor is available as a by-product of their results. We believe this is an interesting open problem that can benefit from the experience on cleaning structured data to introduce the concept of "repairing" natural language sentences.

References

[1] Facebook 'still making money from anti-vax sites'. https://www.theguardian.com/technology/2021/jan/30/facebook-letting-fake-news-spreaders-profit-investigators-claim.

[2] Managing the covid-19 infodemic. https://www.who.int/news/item/23-09-2020-managing-the-covid-19-infodemic-promoting-healthy-behaviours-and-mitigating-the-harm-from-misinformation-and-disinformation.

[3] One of the internet's oldest fact-checking organizations is overwhelmed by coronavirus misinformation - and it could have deadly consequences. https://www.businessinsider.fr/us/coronavirus-snopes-misinformation-fact-checking-overwhelmed-deadly-consequences-2020-3.

[4] Rishabh Agarwal, Chen Liang, Dale Schuurmans, and Mohammad Norouzi. Learning to generalize from sparse and underspecified rewards. *CoRR*, abs/1902.07198, 2019.

[5] Naser Ahmadi, Joohyung Lee, Paolo Papotti, and Mohammed Saeed. Explainable fact checking with probabilistic answer set programming. In *Conference for Truth and Trust Online (TTO)*, 2019.

[6] João Pedro Baptista and Anabela Gradim. Understanding fake news consumption: A review. *Social Sciences*, 9(10), 2020.

[7] Max Bartolo, Tristan Thrush, Robin Jia, Sebastian Riedel, Pontus Stenetorp, and Douwe Kiela. Improving question answering model robustness with synthetic adversarial data generation. *CoRR*, abs/2104.08678, 2021.

[8] Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. Tabfact : A large-scale dataset for table-based fact verification. In *ICLR*, April 2020.

[9] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*, pages 458–469. IEEE Computer Society, 2013.

[10] Artur d'Avila Garcez and Luís C. Lamb. Neurosymbolic AI: the 3rd wave. *CoRR*, abs/2012.05876, 2020.

[11] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proc. of the NAACL*, 2019.

[13] Naeemul Hassan, Fatma Arslan, Chengkai Li, and Mark Tremayne. Toward automated fact-checking: Detecting check-worthy factual claims by claimbuster. In *KDD*, 2017.

[14] Naeemul Hassan, Gensheng Zhang, Fatma Arslan, Josue Caraballo, Damian Jimenez, Siddhant Gawsane, Shohedul Hasan, Minumol Joseph, Aaditya Kulkarni, Anil Kumar Nayak, Vikas Sable, Chengkai Li, and Mark Tremayne. Claimbuster: The first-ever end-to-end fact-checking system. *Proc. VLDB Endow.*, 10(12):1945–1948, August 2017.

[15] Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. TaPas: Weakly supervised table parsing via pre-training. In *Proc. of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333. ACL, 2020.

[16] Viet-Phi Huynh and Paolo Papotti. A benchmark for fact checking algorithms built on knowledge bases. In *Proc. of CIKM 2019, Beijing, China, November 3-7, 2019*, pages 689–698. ACM, 2019.

[17] Alon Jacovi and Yoav Goldberg. Towards faithfully interpretable NLP systems: How should we define and evaluate faithfulness? In *Proc. of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4198–4205, Online, July 2020. Association for Computational Linguistics.

[18] Saehan Jo, Immanuel Trummer, Weicheng Yu, Xuezhi Wang, Cong Yu, Daniel Liu, and Niyati Mehta. Verifying text summaries of relational data sets. In *SIGMOD*, page 299–316. ACM, 2019.

[19] Georgios Karagiannis, Mohammed Saeed, Paolo Papotti, and Immanuel Trummer. Scrutinizer: Fact checking statistical claims. *Proc. VLDB Endow.*, 13(12):2965–2968, August 2020.

[20] Nayeon Lee, Belinda Z. Li, Sinong Wang, Wen-tau Yih, Hao Ma, and Madian Khabsa. Language models as fact checkers? In *Proceedings of the Third Workshop on Fact Extraction and VERification (FEVER)*, pages 36–41, Online, July 2020. Association for Computational Linguistics.

[21] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, and Christian Bizer. Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 6, 01 2014.

[22] Bill MacCartney and Christopher D. Manning. *Natural Logic and Natural Language Inference*, pages 129–147. Springer Netherlands, Dordrecht, 2014.

[23] Sewon Min, Danqi Chen, Hannaneh Hajishirzi, and Luke Zettlemoyer. A discrete hard EM approach for weakly supervised question answering. In *EMNLP-IJCNLP*, pages 2851–2864. ACL, 2019.

[24] Thomas Mueller, Francesco Piccinno, Peter Shaw, Massimo Nicosia, and Yasemin Altun. Answering conversational questions on structured data without logical forms. In *EMNLP-IJCNLP*, pages 5902–5910. ACL, 2019.

[25] Preslav Nakov, David P. A. Corney, Maram Hasanain, Firoj Alam, Tamer Elsayed, Alberto Barrón-Cedeño, Paolo Papotti, Shaden Shaar, and Giovanni Da San Martino. Automated fact-checking for assisting human fact-checkers. In *IJCAI*, pages 4826–4832. ijcai.org, 2021.

[26] Arvind Neelakantan, Quoc V. Le, Martin Abadi, Andrew McCallum, and Dario Amodei. Learning a natural language interface with neural programmer, 2016.

[27] Yixin Nie, Haonan Chen, and Mohit Bansal. Combining fact extraction and verification with neural semantic matching networks. In *The Thirty-Third AAAI Conference on Artificial Intelligence*, pages 6859–6866. AAAI Press, 2019.

[28] Ehud Reiter and Craig Thomson. Shared task on evaluating accuracy. In *Proceedings of the 13th International Conference on Natural Language Generation*, pages 227–231. ACL, 2020.

[29] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the predictions of any classifier. In *SIGKDD*, KDD '16, page 1135–1144, New York, NY, USA, 2016. ACM.

[30] Kai Shu, Amy Sliva, Suhang Wang, Jiliang Tang, and Huan Liu. Fake news detection on social media: A data mining perspective. *SIGKDD Explor. Newsl.*, 19(1):22–36, September 2017.

[31] James Thorne and Andreas Vlachos. Automated fact checking: Task formulations, methods and future directions. In *Proc. of the 27th International Conference on Computational Linguistics*, pages 3346–3359. ACL, 2018.

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *NIPS*, volume 30. Curran Associates, 2017.

[33] Bailin Wang, Ivan Titov, and Mirella Lapata. Learning semantic parsers from denotations with latent structured alignments and abstract programs. In *EMNLP-IJCNLP*, pages 3774–3785. Association for Computational Linguistics, November 2019.

[34] You Wu, Pankaj K. Agarwal, Chengkai Li, Jun Yang, and Cong Yu. Toward computational fact-checking. *Proc. VLDB Endow.*, 7(7):589–600, March 2014.

[35] Yiben Yang, Chaitanya Malaviya, Jared Fernandez, Swabha Swayamdipta, Ronan Le Bras, Ji-Ping Wang, Chandra Bhagavatula, Yejin Choi, and Doug Downey. Generative data augmentation for commonsense reasoning. In *EMNLP*, pages 1008–1025. ACL, 2020.

[36] Takuma Yoneda, Jeff Mitchell, Johannes Welbl, Pontus Stenetorp, and Sebastian Riedel. UCL machine reading group: Four factor framework for fact finding (HexaF). In *Proc. of the First Workshop on Fact Extraction and VERification (FEVER)*, pages 97–102, Brussels, Belgium, November 2018. ACL.

[37] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2SQL: Generating structured queries from natural language using reinforcement learning, 2018.