Fact Checking Machine Generated Text with Dependency Trees

Alex Estes[*] [†]
Balto, Seattle, WA
g.alex.estes@gmail.com

Nikhita Vedula[*], **Marcus Collins**[*], **Matthew Cecil,** and **Oleg Rokhlenko**
Amazon, Seattle, WA
{veduln,collmr,mattceci,olegro}@amazon.com

Abstract

Factual and logical errors made by Natural Language Generation (NLG) systems limit their applicability in many settings. We study this problem in a conversational search and recommendation setting, and observe that we can often make two simplifying assumptions in this domain: (i) there exists a body of structured knowledge we can use for verifying factuality of generated text; and (ii) the text to be factually assessed typically has a well-defined structure and style. Grounded in these assumptions, we propose a fast, unsupervised and explainable technique, DepChecker, that assesses factuality of input text based on rules derived from structured knowledge patterns and dependency relations with respect to the input text. We show that DepChecker outperforms state-of-the-art, general purpose fact-checking techniques in this special, but important case.

1 Introduction and Background

Prior work has noted the benefits of natural language text generated by NLG models over fixed templates, for various language processing and language understanding tasks, i.e., fewer grammatical or structural disfluencies, increased conversational nature, textual diversity and user satisfaction (Kale and Rastogi, 2020; Challa et al., 2019; Vedula et al., 2023). Recent NLG approaches based on language models allow conversational agent designers to create natural-sounding responses in a very wide range of situations without exhaustively testing template-based response text (Wang et al., 2022; Asai et al., 2021; Kim et al., 2021). However, neural NLG systems may emit hallucinated, factually incorrect or even nonsensical content that is not entailed by their inputs (Wang et al., 2020; Tian et al., 2019; Liu et al., 2021b; Dhingra et al., 2019). For instance, an NLG model may synthesize multiple independent hypotheses involving comparisons or aggregations, which must be verified. There has been growing attention towards checking the factual accuracy of statements produced by neural NLG systems in conversational settings. These statements fall into two categories depending on the source input data used for generation (Guo et al., 2022). First, the input whose factuality is to be determined can be *unstructured*, as in abstractive summarization (Teredesai et al., 2019; Goyal and Durrett, 2020) or the FEVER task (Thorne et al., 2018; Zhou et al., 2019; Zhang et al., 2020; Bekoulis et al., 2021; Vedula and Parthasarathy, 2021). Second, the input to be factually assessed may involve *structured* components, like knowledge graphs (Auer et al., 2007; Shiralkar et al., 2017) or tabular data (Chen et al., 2019; Zhong et al., 2020; Liu et al., 2021a).

In conversational, high-consequence domains like health, finance, or e-commerce, we must be sure that any factual error introduced by an NLG system is detected (Chen et al., 2021; Di Sotto and Viviani, 2022; Khan et al., 2022). While superficially similar to other fact verification efforts, this setting presents both unique challenges, and unique simplifications. In this work, we investigate the factuality of fluent e-commerce statements generated via neural NLG techniques using structured data, namely, a commercial product catalog (Ni et al., 2019) containing tables of products and their attributes. We show that the state-of-the-art systems typically developed for general-purpose structured data fail to perform well in our conversational e-commerce setting. Though we focus on the e-commerce domain, our proposed system can be used with any structured data source to check statements generated from non-e-commerce domains such as finance (e.g., the recent FinQA challenge (Chen et al., 2021) or health (consider the consequences of providing inaccurate health recommendations to users of personal health trackers

[*] These three authors contributed equally
[†] Work done while at Amazon

in a system like (Harris and Zaki, 2022)).

We focus on three main challenges while building our proposed fact checking model, *DepChecker*. First, since presenting factually untrue statements to users in high-consequence domains may lead to a poor decision, we must achieve very high false statement recall, and very high true statement precision. Second, deploying fact verification in a conversational setting demands speed – during runtime evaluation of responses to customers of voice assistants, and in case we want to use fact verification output as a signal to improve an accompanying conversational response generation model. As a result, we seek to avoid using large language models for fact checking in production, if possible. Third, data-to-text NLG systems may rephrase entities from the input values, e.g., if a data entity is itself disfluent. In benchmark corpora like TabFact (Chen et al., 2019), data and reference text mentions of the same entity are generally identical, which can limit the flexibility of fact verification models trained on those corpora.

DepChecker also benefits from some simplifications, especially in the e-commerce domain. We observe that in the conversational product search and recommendation setting, NLG systems are unlikely to generate product statements that have a complicated linguistic style or form (Jannach et al., 2021). Thus, unlike TabFact which requires complex hops and joins to reason over structured table data, we can concentrate on a simpler, reduced set of operations. We are inspired by (Reddy et al., 2016), who approached question answering by developing rules over dependency parse trees. Consider the example in Figure 1: key entities have very simple structures. The product mention has an `nsubj` (nominal subject) dependency[1]; attributes are either `attr` (attribute) or `conj` (conjunct) dependents of `attr`. These basic observations hold over a wide set of examples in this domain, prompting us to exploit dependency trees to better extract claims, perform entity linking to match the input statement with its associated structured data, and finally determine the factuality of the statement.

A current limitation of DepChecker could be that manually constructing rules over parse trees requires both time and domain expertise. As a result, we also seek to automatically induce the manually generated rules using genetic optimization algorithms (Mitchell, 1998), in a system we call DepCheckerGA. Our contributions can thus be summarized as follows:

1. We develop a high-speed fact-verification system that has a very high false statement recall and very high true statement precision, for future deployment in production.
2. We show that highly performing fact verification models on structured, open domain data benchmarks fail to perform well on NLG statements created from structured, domain-specific inputs, despite fine-tuning.
3. We show that our unsupervised, accurate system DepChecker is explainable and up to $10\times$ faster than neural baselines.
4. We discuss how DepChecker's rules can be automatically learned and generalized to other systems or domains via genetic optimization.

2 Methodology

2.1 Parsing and Fact Extraction Rules

DepChecker consists of three high-level components: (1) a set of rules applied to a dependency parse to identify the heads of entity mentions; (2) rules over token tags and text used to identify the complete mention; and (3) a system to link the identified entity mentions to corresponding structured evidence, and thereby verify whether the text is factual. Here, we focus on the product names, attribute names, and attribute values that are part of our structured product catalog. We use `spaCy`'s dependency parser[2] to extract product-attribute relations from the text. We describe several examples of rule development in Section 2.2. These rules however only identify the head-tokens of product and attribute mentions. A second set of patterns over exact strings, regular expressions, lemmas, parts of speech, and other linguistic attributes is used to complete the entity spans, described in Section 2.3. We end with product-attribute-value relations like `{product_3, size, 550w}` (see Figure 1), which we call a "hypothesis". We note that an NLG based text generation model may produce non-standard statements which the parser may not parse "correctly". We emphasize that DepChecker is not actually affected by the correctness of the dependency parse, so long as it is *consistent* across all statements in the domain. We are using the parse

[1] We use ClearNLP dependency labels: github.com/clir/clearnlp-guidelines

[2] `spacy.io/usage/linguistic-features`

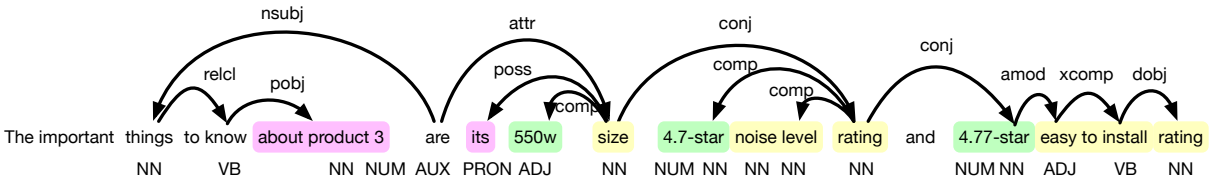

Figure 1: Identifying entity attributes from the dependency parse tree of an input claim whose factuality is to be assessed. Potential product mentions are shown in pink, attribute names in yellow, and attribute values in green.

only as a set of features to identify product and attribute mentions and their relations to each other.

Statements in this shopping domain can also compare attributes of different products. For example, for the claim "With 5 stars, Product 1 has the highest battery life rating of all three products" to be true, Product 1's battery life rating must be 5 stars, and the other products' battery life ratings must be less than 5 stars. We add to the hypotheses one or more *comparands* (products to which we compare), and a *comparator* (*e.g.*, "more" or "less"). Here, there are two implied pairwise comparisons, one each with Products 2 and 3.

The third component evaluates the extracted hypotheses against the catalog data. For each hypothesis extracted from a claim, we iterate through the catalog attributes to find a match. We first check for an exact match. If there is none, we search for catalog and hypothesis attributes whose cosine similarity of their GloVe embeddings (Pennington et al., 2014) is more than 0.9. If any hypothesis fails to match, then the claim is False. If all hypotheses evaluate to True, the claim is labeled True.

2.2 Dependency Rule Development

We illustrate the process of finding good rules to identify product-attribute relationships with two examples. We annotate 250 examples and identify the dependency paths from key verbs (*e.g.*, "has", "is") and possessives (`poss`) to product attributes. For instance, the `dobj` (direct object) of "has" is always a product attribute phrase. An obvious example is "Product 1 has four processors". Now consider the example in Figure 1. We identify the verb *are*, and descend the tree from there to identify products and attributes. The first attribute ("550w size") has a `attr` dependency arc from "are". Remaining attributes are all (`conj`) children of another attribute. DepChecker uses six of these *attribute existence rules* rules to extract attributes.

Figure 2 shows a case with a possessive pronoun, *its*, whose head, *quality*, is universally an attribute,

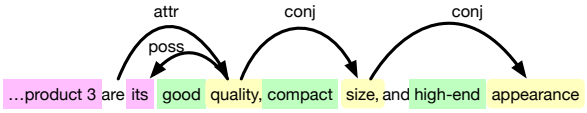

Figure 2: An example parse with a possessive, showing that attributes are the head, and conjunct dependencies of the head, of *its*. Color scheme as in Figure 1.

so we add a rule `head(poss)`→`attribute`. Again, `conj` arcs link remaining attributes to the first. We follow a similar process to find the product mention, first climbing the tree to the main verb of the sentence (here "like", not shown in the figure), and descending again to find the product mention. DepChecker has six rules for possessives. Two additional rules handle more generic verb heads, to identify product attributes phrased like "the lights are bright". DepChecker has six more rules to handle comparisons, closely following attribute existence rules. Thus, DepChecker relies on just twenty rules to identify products from dependency trees.

2.3 Attribute Token Rules

Our dependency rules find the head token of a product or attribute, and we must identify the complete attribute and value, if present, for which we developed regular expressions on the part-of-speech tags. In all, there are 30 rules for different products and attributes. For example, to extract the complete attribute "easy to set up", we use the pattern `/(NN|ADV)* ADJ (ADP|PART) (VB|NN) NN? ADP?/`. Finally, taking advantage of the relatively simple statement structure for the e-commerce domain, we have two co-reference rules linking pronouns to product mentions. We observe that these rules are more accurate and faster than co-reference resolution with general purpose models (Stylianou and Vlahavas, 2021).

2.4 Genetic Rule Optimization

We now describe how we can automatically learn the rules detailed in the above sections. To extract

product:attribute and *attribute name:value* pairs from statements, we first observe that we can identify a path from any token in the sentence up to a common head and down to any other token. Each pair of paths from two tokens to a shared head can be expressed as a list of dependencies. For instance, in Figure 1, "product" has a path `[pobj, relcl, nsubj]` to the shared head token "are", while the attribute phrase "550w size" has a path `[attr]` to "are". Together, these paths form a candidate rule for *product:attribute* extraction.

We extract all such candidate rules from a small set of ground truth-labeled example sentences. In principle, we could apply all of the extracted rules, but not all are sufficiently precise. We therefore apply genetic optimization (Fortin et al., 2012; Mitchell, 1998), optimizing our rule selection to increase product and attribute phrase head recall, while maximizing token level precision to avoid selecting spurious phrases.

We start from a set of candidate rules R_{cand}, which are regular expressions over the dependency labels linking products and attributes to common head tokens. We further define a fitness function E which we use to optimize individuals I_i, which are collections of rules, and the population P as a whole. In our case, we optimize for a combination of precision and recall in identifying labeled products and attributes, and add a penalty for adding more rules. To initialize the population, we create n_{pop} individuals, each of which is initialized by randomly selecting between 1 and 8 rules from R_{cand}. Next, we begin to evolve the population. At each step, we evaluate each individual in the population using E. While our best individual is below some target threshold fitness E_{tgt}, we select individuals, mutate some of them, and recombine rules between individuals.

Selection consists of choosing the most "fit" individuals from the population, based on the fitness evaluated using E. We use tournament selection (Mitchell, 1998), implemented using `deap` (Fortin et al., 2012)[3]. In tournament selection, we choose a tournament size k (10 in our case) and select that number participants randomly from the current population generation. The most fit individual from that subset of k individuals is added to the next generation. We use sampling with replacement, so the most fit individuals are likely to be added more than once to the next generation, strengthening the next generation while also allowing for some variation to carry forward.

Next, we iterate over all individuals, first removing from each individual any redundant rules which are wholly entailed by some other rule. Then with probability p_{mut}, we mutate the individual, and recombine or "crossover" rules with another individual with probability p_{cr}. We mutate individuals to diversify the new rule generation. Then, we merge rules. Merging consists of aligning pairs of rules and combining the aligned forms. We use the Needleman-Wunsch (Needleman and Wunsch, 1970) algorithm to align the patterns. We choose the most closely aligned pairs of rules to merge. Where the tokens match, we simply add them to the pattern; where there are gaps, we add a '`*`' (repeater or Kleene star) operator, and where there are differences, we use an '`|`' (or) operator. For example, given the patterns `ABCD` and `ACE`, the alignment and merge would result in `AB*C(D|E)`. As we will explain below, this pairwise alignment is somewhat limited, and can lead to complex patterns that are hard to further combine or generalize.

If we recombine rules with another individual, we first select an individual at random. We then choose a rule at random from each individual, and swap them between the two individuals.

3 Evaluation

To test our proposed approaches, DepChecker and DepCheckerGA, we first developed a BART-based (Lewis et al., 2020) data-to-text NLG model to generate conversational statements in the e-commerce domain, taking as input tabular product-attribute data from a structured product catalog (Ni et al., 2019). To train this NLG model, we asked human annotators to write text describing and comparing the product attribute values of 2-3 catalog products (Figures 1 and 2 contain examples of the human generated sentences). We used this set of NLG model generated sentences as our test set for factuality assessment, to test DepChecker in this work. As a baseline for DepChecker, we also trained a supervised factual error detection binary classifier (henceforth called RoBERTaChecker) using the RoBERTa (Liu et al., 2019) language model, with a mix of the above mentioned human-generated product domain statements and artificially generated negative examples (generated by corrupting the entity names and values in the human-generated factually correct statements). This binary classifi-

[3] `deap.readthedocs.io`

Method	Acc	TP	FR	AIT
Table-BERT (Chen et al., 2019)	0.56	0.50	0.31	452
GNN-TabFact (Ran et al., 2019)	0.43	0.53	0.20	644
TAPEX-base (Liu et al., 2021a)	0.52	0.54	0.79	228
TAPAS-base (Herzig et al., 2020)	0.51	0.66	0.97	724
RoBERTaChecker	0.67	0.87	0.94	681
DepChecker (proposed)	0.65	0.85	0.94	72
DepCheckerGA (proposed)	0.63	0.84	0.94	256

Table 1: Fact verification performance of different models. **Acc**: Accuracy; **TP**: True precision; **FR**: False recall; **AIT**: Average Inference Time (in milliseconds).

cation dataset was also used to fine-tune the neural baselines shown in Table 1. The first four baselines are approaches that have been specifically developed for the TabFact dataset (Chen et al., 2019), and were also fine-tuned on TabFact. We also labeled some of this data with product name, attribute name, and attribute value tags, to select dependency rules as described in Section 2.4.

The factual error detection test set to evaluate DepChecker consists of 195 sentences generated by our BART based NLG model described above, and includes 98 factually consistent or true statements and 97 factually inconsistent or false statements. This test set was manually labeled after finalizing the DepChecker rules, to avoid any data leakage. The values of all hyperparameters used by DepChecker and DepCheckerGA were tuned on a manually curated validation set. We report the average execution time over all examples at inference, on a single 2.3GHz Intel Xeon E5-2686 v4 CPU (an Amazon AWS EC2 p3.8xlarge instance). Note that we explicitly include the dependency parsing time in the reported execution time for DepChecker. An example of a structured product table from our dataset is `{size_product3: 550w, noise level rating_product3: 4.7, easy to install rating_product3: 4.77}`, from which our BART based NLG model generates the sentence shown in Figure 1.

3.1 Results

As shown in Table 1, DepChecker outperforms all baselines on accuracy or speed, and outperforms most models on both. Models developed for large-scale fact checking baselines have surprisingly low accuracy: the best system, TAPAS (Herzig et al., 2020) reaches only 51%: it labels nearly all examples as false. A more important task here is to detect false statements, since making a false statement to a customer in an e-commerce setting could seriously damage a business's reputation. TAPAS identifies 97.0% of factually inconsistent sentences (at the cost of very low precision on factually true statements), followed by our baseline neural model, RoBERTaChecker, with false recall 94% but much better overall accuracy 67%. DepChecker is equally good, reaching 94% false recall and 65% accuracy. In addition, neural models, especially those using transformer architectures, are slow. Using GPUs makes them faster, but much more costly. For both cost and energy impact in a production deployment setting, as well as for a lower latency in a conversational response generation setting, it is desirable to use more efficient models. Table 1 shows that DepChecker is three to ten times faster than all other baseline models, and 9.4 times faster than the only neural model with comparable accuracy, RoBERTaChecker.

We next test how well we could automatically reproduce DepChecker's rules using our genetic optimization scheme in Section 2.4 (*DepCheckerGA*). We find that the rules generated by genetic optimization are not as compact or efficient as manually created rules. These rules extract 95% of head tokens of the 1000 product name, attribute name, and attribute value spans in our test set, and 96% of the extracted tokens belonging to those spans. These values are statistically indistinguishable from DepChecker, albeit with almost twice as many rules. As a result of the additional rules, average execution time increases to about 250 milliseconds per sentence[4], making DepCheckerGA still superior to several of our baseline systems.

3.2 Discussion

Error Analysis and Explainability: Of all models, DepChecker and RoBERTaChecker have significantly higher accuracy and high false recall. They mis-classify very few false statements, and also make different errors. Combined, these two models achieve 99% false recall, better than any other system. RoBERTaChecker makes errors on encountering attributes altered from the input table ("80 plus gold" → "80 plus gram"), misses negation (attribute "heat sensitive: false" → "customers like its heat sensitivity", or when an attribute value is associated with the wrong product. DepChecker makes errors only of the first kind, *e.g.*, if it gets as input "soft material" for the attribute "soft".

Other neural models appear to make similar er-

[4] Note that to implement our genetic optimization, we convert dependency parses to strings and execute regular expression rules; it may be possible to further optimize this by converting the rules to SpaCy's tree regex, used in DepChecker.

rors to RoBERTaChecker. For instance, TAPEX classifies "... product 3's 1.0-inch unit count ...", as true even though it is obviously wrong; DepChecker classifies it correctly. TAPEX and TAPAS both fail in cases where NLG transforms attributes like "battery" to "battery life", or when the NLG fabricates an attribute completely, with no corresponding input value. As with RoBERTaChecker, TAPEX and TAPAS make orthogonal errors to DepChecker, and near 100% false recall can be achieved. We aren't sure why this should be, since the neural models are opaque. However, unlike other state-of-the-art systems, our proposed approach DepChecker is both fast and explainable. Its rules make it clearer to diagnose and understand the exact reason DepChecker fails on an input, and use this signal to correct the input claim or improve the fact verification process. DepChecker's speed also enables it to be easily used in real-time systems, and as a potential signal to improve the training of NLG models generating factually inconsistent statements.

Comparing DepChecker and DepCheckerGA rules: We assess whether DepCheckerGA can learn similar rules as those created by humans or expert linguists (see Table 2). First, we observe DepCheckerGA's rules tend to be more narrow, illustrated by the second example in Table 2, which has a number of additional constraints compared to the DepChecker rule, which has broader coverage as well. This is likely due to the fact that in order to facilitate the genetic optimization strategy, DepCheckerGA must fully specify the dependency paths to each product and attribute in its rules. By contrast, a linguist can find patterns that involve only the key parts of the dependency links. It may also be partly due to the limitations of our alignment and merging strategy in Section 2.4. This leads to more, and unnecessarily specific rules.

In many cases, however, the rules found by DepCheckerGA match those formed by expert linguists. The first and third examples in Table 2 demonstrate this. However, we note that DepCheckerGA has redundant rules that identify different parts of the same sentences. In the third example, the rules shown find a link between "product", "strap", and the shared head "style". Another DepCheckerGA rule, NOUN `conj* ((pobj prep)|amod)`[5], unnecessarily identifies "adjustable", showing that DepCheckerGA could also do a better job pruning its rules.

Finally, while matches are more or less uniformly distributed across our hand-crafted rules, only about 1 in 6 of DepCheckerGA's rules match more than 10 statements. In future, we will further explore DepChecker's rule combining strategies for it to match DepChecker's performance. Further, our rule merging relies on two rules being mostly identical. We do not merge multiple rules at once to generalize better. Multiple sequence alignment techniques could be used to improve DepCheckerGA (Chowdhury and Garai, 2017).

Dependency Parsing Errors: As noted in Section 2.1, since we learn dependency patterns from the parsed NLG text, the parse need only be *consistent* across the domain, and not linguistically correct. Parsing errors are taken into account while learning the rules of DepChecker. It thus implicitly handles any language errors made by the chosen dependency parser, which might be more common if the NLG input is not completely fluent. For instance, in the statement *customers feel highly positive about product 1's great features*, the parser incorrectly labels "features" as the object of adposition *about*, when it should be *product*. Both DepChecker and DepCheckerGA learn rules to handle this technical error.

Generalizability: We showed that we can automatically learn a comparable set of rules over dependency parse trees using genetic optimization (Section 2.4). This allows DepChecker to generalize to other domains, requiring only a small set of data instances labeled with the relevant entity names and values. It also allows a domain-specific fact verification system to be built rapidly with almost no manual effort. However, as mentioned earlier, the automatically learned rules may be less efficient, so some expert linguist effort may still be required. Our analysis indicates that our domain of conversational shopping likely exhibits a comparatively small amount of variation in style and linguistic structure of the text whose factuality is being assessed in this paper. Extending DepChecker to work well for domains with higher textual variation is something we leave for future work.

To investigate DepChecker's generalizability to large-scale, general purpose corpora like TabFact, we extend our defined rules to a sample of TabFact entities, which have been derived from DBPedia (Auer et al., 2007). We replace the 'product' and 'attribute' entity mentions in our rules by more

[5] see Table 2 caption for notation

	attribute rule; product rule *or* example sentence
DepChecker	*have* **dobj**; *have* nsubj
DepCheckerGA	*have* **dobj**; *have* nsubj conj*
example sentence	Product 2 has the highest **rating**.
DepChecker	*have* **dobj** .* **conj**; *have* nsubj
DepCheckerGA	*have* **dobj**; VB advcl (acomp|dobj*) **conj** (acomp|amod*)*
example sentence	According to reviews, previous buyers feel positive about product 1 because it has easy setup and **installation**, great picture **quality**, and lights are **bright**.
DepChecker	*attribute product* case; *attribute* .* **conj**
DepCheckerGA	NOUN **conj***; NOUN (nmod|poss)
example sentence	previous buyers feel highly positive about product 1's laptop style, adjustable **strap**, and 17.3-inch grey **color**.

Table 2: Comparing similar rules from DepChecker (manually created) and DepCheckerGA (automatically learned via genetic optimization). Rules are expressed as regular expressions over dependency paths. Italics indicates a specific lemma must be at that position. Capital letters indicate a part-of-speech class. Heads are to the left. We use universal dependencies and standard regular expression notation: * means zero or more instances of the token, '.' means any token. Bolded words indicate matched attributes; underlined words indicate matched heads. Note that *attribute* stands for any word, and *product* is a specialized lemma.

generic 'table-row' and 'table-column' tokens of the TabFact tables, and verify if a claim is true, given an input table. DepChecker cannot yet handle some of TabFact's aggregations, so we check the statement veracity for each table row independently. If a claim is false for any one row of the input table, we label it False, otherwise it is labeled True. This simple technique achieves 60% accuracy on the TabFact test set (current state-of-the-art model accuracy is 85%), suggesting the potential of using dependency parse structure in more general or complex fact checking scenarios.

4 Limitations

The major limitation of our work is that DepChecker in its basic form requires manual effort to construct rules for fact checking. While we showed that this rule construction can be automated, it comes with a trade-off in the compactness of the automatically generated rules and the overall latency. However, as shown in Sections 3.1 and 3.2, large, neural language models developed for fact checking with structured general purpose data (e.g., TabFact derived from Wikipedia tables) are unable to perform fact checking well on statements based on our e-commerce structured catalog data. The time required to assemble such a large volume of labeled data in specific domains (e.g., health or e-commerce) to train large neural models would certainly cost more than one or two weeks of a linguist's time, and may still not be fast or accurate enough to be deployed for production purposes. Hence, it isn't clear how much effort would be required to build a neural system comparable in speed and accuracy to DepChecker, or that the benchmark systems scale or generalize better than DepChecker for a real-world deployment scenario. Like existing data-to-text fact checking work, DepChecker also assumes that the underlying structured data used to verify the statement factuality is accurate.

Finally, we acknowledge that rule-based approaches are not new, but as we show in this work, they can still outperform or at least complement state-of-the-art language models. Their speed, simplicity, accuracy, and explainability make them valuable tools in production use cases. Since seemingly state-of-the-art results based on large language models may not generalize as well as hoped across different domains and datasets, we would like to point out that our rule-based approach, DepChecker, remains competitive in many scenarios. Such a system can therefore still be worth investing in, especially in an age of larger and slower models.

5 Conclusion

We propose DepChecker: a fast, unsupervised, and explainable model to detect factual inconsistencies in a conversational shopping and recommendation setting. We show that DepChecker can outperform strong baselines, and suggest that instead of pursuing costlier, less explainable, slower models, more research should go into how to further automate development of simpler, less expensive and more interpretable models for production use cases.

References

Akari Asai, Matt Gardner, and Hannaneh Hajishirzi. 2021. Evidentiality-guided generation for knowledge-intensive nlp tasks. *arXiv preprint arXiv:2112.08688*.

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer.

Giannis Bekoulis, Christina Papagiannopoulou, and Nikos Deligiannis. 2021. A review on fact extraction and verification. *ACM Comput. Surv.*, 55(1).

Ashwini Challa, Kartikeya Upasani, Anusha Balakrishnan, and Rajen Subba. 2019. Generate, filter, and rank: Grammaticality classification for production-ready nlg systems. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Industry Papers)*, pages 214–225.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2019. Tabfact: A large-scale dataset for table-based fact verification. In *International Conference on Learning Representations*.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. 2021. FinQA: A dataset of numerical reasoning over financial data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3697–3711, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Biswanath Chowdhury and Gautam Garai. 2017. A review on multiple sequence alignment from the perspective of genetic algorithm. *Genomics*, 109(5-6):419–431.

Bhuwan Dhingra, Manaal Faruqui, Ankur Parikh, Ming-Wei Chang, Dipanjan Das, and William Cohen. 2019. Handling divergent reference texts when evaluating table-to-text generation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4884–4895.

Stefano Di Sotto and Marco Viviani. 2022. Health misinformation detection in the social web: An overview and a data science approach. *International Journal of Environmental Research and Public Health*, 19(4):2173.

Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175.

Tanya Goyal and Greg Durrett. 2020. Evaluating Factuality in Generation with Dependency-level Entailment. *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3592–3603.

Zhijiang Guo, Michael Schlichtkrull, and Andreas Vlachos. 2022. A survey on automated fact-checking. *Transactions of the Association for Computational Linguistics*, 10:178–206.

Jonathan Harris and Mohammed J. Zaki. 2022. Towards neural numeric-to-text generation from temporal personal health data. *arxiv*.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Mueller, Francesco Piccinno, and Julian Eisenschlos. 2020. Tapas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333.

Dietmar Jannach, Ahtsham Manzoor, Wanling Cai, and Li Chen. 2021. A survey on conversational recommender systems. *ACM Computing Surveys (CSUR)*, 54(5):1–36.

Mihir Kale and Abhinav Rastogi. 2020. Template guided text generation for task-oriented dialogue. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6505–6520.

Suleman Khan, Saqib Hakak, N Deepa, Kapal Dev, and Silvia Trelova. 2022. Detecting covid-19 related fake news using feature extraction. *Frontiers in Public Health*, page 1967.

Jinhyeon Kim, Donghoon Ham, Jeong-Gwan Lee, and Kee-Eung Kim. 2021. End-to-end document-grounded conversation with encoder-decoder pre-trained language model. In *Proceedings of the DSTC9 Workshop, Online*, pages 8–9.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2021a. Tapex: Table pre-training via learning a neural sql executor. In *International Conference on Learning Representations*.

Tianyu Liu, Xin Zheng, Baobao Chang, and Zhifang Sui. 2021b. Towards faithfulness in open domain table-to-text generation from an entity-centric view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 13415–13423.

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*.

Melanie Mitchell. 1998. *An Introduction to Genetic Algorithms*. MIT Press.

Saul B. Needleman and Christian D. Wunsch. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453.

Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 188–197.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Qiu Ran, Yankai Lin, Peng Li, Jie Zhou, and Zhiyuan Liu. 2019. Numnet: Machine reading comprehension with numerical reasoning. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2474–2484.

Siva Reddy, Oscar Täckström, Michael Collins, Tom Kwiatkowski, Dipanjan Das, Mark Steedman, and Mirella Lapata. 2016. Transforming Dependency Structures to Logical Forms for Semantic Parsing. *Transactions of the Association for Computational Linguistics*, 4:127–140.

Prashant Shiralkar, Alessandro Flammini, Filippo Menczer, and Giovanni Luca Ciampaglia. 2017. Finding streams in knowledge graphs to support fact checking. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 859–864. IEEE.

Nikolaos Stylianou and Ioannis Vlahavas. 2021. A neural entity coreference resolution review. *Expert Systems with Applications*, 168:114466.

Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, George Karypis, Ben Goodrich, Vinay Rao, Peter J Liu, and Mohammad Saleh. 2019. Assessing The Factual Accuracy of Generated Text. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 166–175.

James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. 2018. FEVER: a large-scale dataset for fact extraction and VERification. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 809–819, New Orleans, Louisiana. Association for Computational Linguistics.

Ran Tian, Shashi Narayan, Thibault Sellam, and Ankur P Parikh. 2019. Sticking to the Facts: Confident Decoding for Faithful Data-to-Text Generation. *arXiv*.

Nikhita Vedula, Marcus Collins, Eugene Agichtein, and Oleg Rokhlenko. 2023. Generating explainable product comparisons for online shopping. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*.

Nikhita Vedula and Srinivasan Parthasarathy. 2021. Face-keg: Fact checking explained using knowledge graphs. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 526–534.

Weizhi Wang, Zhirui Zhang, Junliang Guo, Yinpei Dai, Boxing Chen, and Weihua Luo. 2022. Task-oriented dialogue system as natural language generation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2698–2703.

Zhenyi Wang, Xiaoyang Wang, Bang An, Dong Yu, and Changyou Chen. 2020. Towards Faithful Neural Table-to-Text Generation with Content-Matching Constraints. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1072–1086.

Wenxuan Zhang, Yang Deng, Jing Ma, and Wai Lam. 2020. AnswerFact: Fact checking in product question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2407–2417, Online. Association for Computational Linguistics.

Wanjun Zhong, Duyu Tang, Zhangyin Feng, Nan Duan, Ming Zhou, Ming Gong, Linjun Shou, Daxin Jiang, Jiahai Wang, and Jian Yin. 2020. LogicalFactChecker: Leveraging logical operations for fact checking with graph module network. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6053–6065, Online. Association for Computational Linguistics.

Jie Zhou, Xu Han, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2019. GEAR: Graph-based evidence aggregating and reasoning for fact verification. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 892–901, Florence, Italy. Association for Computational Linguistics.