WebChecker: Towards an Infrastructure for Efficient Misinformation Detection at Web Scale

Immanuel Trummer
Cornell University
itrummer@cornell.edu

Abstract

We focus on scaling up fact checking to very large document collections. Given a computational cost budget, our goal is to find a maximal number of instances of misinformation. We present the WebChecker, a platform that leverages indexes, cheap filters, and matching methods with various cost-accuracy tradeoffs to maximize fact checking efficiency. It uses a reinforcement learning based optimizer to find optimal checking plans. In our experiments, we use an early prototype to find misinformation on the Web, exploiting Google search to retrieve Web documents and pre-trained language models to identify problematic text snippets within them. WebChecker finds significantly more matches per time unit, compared to naive baselines, and reliably identifies near-optimal checking plans within its plan space.

1 Introduction

Misinformation on the Web can have disastrous consequences [32]. Major Web companies such as Google, Facebook, and Twitter have recently taken steps to protect their user base from being served misleading content [17, 44]. However, those efforts still rely significantly on teams of human fact checkers who are all too often overwhelmed by the scale at which misinformation propagates [36, 46]. Over the past years, this has motivated significant research on automated fact checking methods [10], in industry as well as in academia.

At its core, automated fact checking requires analyzing natural language text (for instance, in order to verify whether a given claim is equivalent to a previously verified one). The area of natural language processing (NLP) has recently seen significant progress. In particular, large pre-trained models based on the Transformer [43] architecture have advanced the state of the art in multiple sub-areas of NLP [16, 47] and achieve near human-level performance for various NLP tasks [5, 28].

The increased accuracy of recent NLP approaches comes, however, at a price. State-of-the-art performance often requires large neural networks with hundreds of millions [7] to hundreds of billions of parameters [9]. Inference via such models is costly. Applying such models naively to large collections of documents (for fact checking or other tasks) is prohibitively expensive. Prior work on automated fact checking has often focused on verifying single claims. Here, given methods to verify single claims, we are interested in scaling up automated checking to large collections of documents, even up to Web scale. In order to do so, we leverage classical techniques from the database community, in particular query planning [37] and adaptive processing [3].

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

We describe and evaluate a first prototype of the WebChecker, a platform aimed at misinformation detection in very large document collections. Of course, finding all mistakes in a sufficiently large set of documents (say, the Web) is not realistic. Hence, our goal is to find the maximal number of instances of misinformation under a fixed, computational budget instead. As an extension, we may assign different weights to different instances of misinformation and prioritize search accordingly.

As one possible use case, we envision WebChecker as a means to trigger corrective actions quickly if misinformation is detected. For instance, WebChecker could notify Wikipedia administrators if factual mistakes appear on pages that fall within their core domain. Alternatively, WebChecker could create tags requesting verification by volunteers from the Wikipedia community. Similarly, WebChecker could be applied to verify content on large Web platforms such as Twitter, notifying internal fact checker teams of potentially problematic content (triggering platform-specific counter-measures). Note that new content is generated at a regular pace in all of the aforementioned use cases. This means that fact checking is a continuous process, rather than a one-time operation. This makes computational efficiency, our focus, key.

The WebChecker takes as input a repository containing misinformation and a collection of documents. Its goal is to match text snippets in documents to entries in the repository. To do so efficiently, WebChecker uses a collection of fact checking methods, including high and low precision matching methods, cheap, heuristic filters, and indexes. A-priori, it is unclear which combination of methods maximizes the number of matches found within the cost budget. WebChecker therefore uses an adaptive processing approach, switching between different processing plans to sample their quality. To decide which plans to try next, WebChecker uses reinforcement learning methods.

We evaluate an early prototype of WebChecker experimentally (source code for this prototype is available at `https://github.com/itrummer/WebChecker`). We show that plan choices have significant impact on the number of matches found (up to two orders of magnitude). Furthermore, we show that WebChecker successfully converges to near-optimal plans via reinforcement learning. At the same time, our experimental results point at important avenues for improvements.

The remainder of this paper is organized as follows. We introduce our problem model formally in Section 2. In Section 3, we give a high-level overview of the WebChecker prototype. In Section 4, we describe the space of misinformation detection plans considered by the prototype. Then, in Section 5, we describe the adaptive processing approach used to identify promising plans. We present first experimental results in Section 6. Finally, we discuss related work in Section 7 and conclude with future work plans in Section 8.

2 Problem Model and Terminology

Our goal is to find misinformation in a document collection. We detect misinformation at a fine granularity, verifying specific text snippets (as opposed to classifying entire documents as fake news). We consider large document collections (up to "Web-scale") where verifying each single text snippet naively is prohibitively expensive. Hence our focus on maximizing detection efficiency. To identify misinformation, we compare against entries in a so-called anti-knowledge base [24] (also known as "negative knowledge" [2]), defined next.

Definition 1 (Anti-Knowledge Base, Anti-Fact): An Anti-Knowledge Base (AKB) is a collection A of Anti-Facts, each one representing a piece of information known to be wrong. Each anti-fact $a \in A$ is expressed as a subject-predicate-object triple $a = \langle S, P, O \rangle$. It expresses a relationship between subject and object that does not hold. In addition, each anti-fact may be associated with one or multiple tags (expressing for instance the source from which the anti-fact was mined).

We assume that anti-facts are indexed by tags such that anti-facts with specific tags can be retrieved efficiently. Our goal is to match AKB entries to text snippets in the aforementioned document collection.

Definition 2 (AKB Match): An AKB Match is described as a pair $\langle a, s\rangle$ where $a \in A$ is an anti-fact from the AKB, $s \in D$ is a text snippet (e.g., a single sentence) from the document collection D. Snippet and anti-fact are related in that the text describes the anti-fact, i.e. the text snippet contains misinformation.

Identifying matches between AKB entries and text snippets requires natural language analysis. This field has advanced quickly in recent years, seeing widespread adoption of novel and promising methods such as the Transformer architecture [43]. Nevertheless, analysis precision is still not perfect, leading to erroneous matches.

Definition 3 (Matching Precision, False Positive): With regards to a set of (potential) matches $M = \{\langle a_i, s_i\rangle\}$, we call the ratio of true matches the Matching Precision. We call any pair $\langle a, s\rangle \in M$ where s does not express the anti-fact a a False Positive. We compare different matching methods by their (expected) matching precision.

Setting a different precision target allows adapting WebChecker to different scenarios (e.g., increasing recall at the expense of precision if potential matches are shared with a large crowd for verification while preferring few, high-precision matches if results are forwarded to a small team of fact checkers).

Given many documents and AKB entries, it is often not realistic to find all instances of misinformation. Hence, we must focus our efforts on high-priority instances. To do so, we introduce a weighted recall metric.

Definition 4 (Match Weighting Function): The detection process can be configured by providing a Match Weighting Function. This function assigns each AKB match to a weight, expressing importance of the match. The match weighting function $w_M : M \mapsto \mathbb{R}^+$ assigns AKB matches $m \in M$ to a numerical, positive weight. The weight of a match $\langle a, s\rangle$ is the product of two weight functions, w_A and w_S, assigning a weight for the AKB entry and the text snippet respectively (i.e., $w_M(\langle a, s\rangle) = w_A(a) \cdot w_S(s)$).

The weighting function w_A allows assigning higher weights for misinformation that relates to specific topics (e.g., for anti-facts about the Coronavirus that may accelerate the spread). Weighting function w_S allows assigning higher weights for specific Web sites (e.g., to prioritize highly visible Web sites where misinformation could be particularly harmful). WebChecker prioritizes highly weighted matches in its search. More precisely, it processes detection tasks as defined next.

Definition 5 (Detection Task, Detection Result): A Detection Task is defined by a tuple $\langle A, D, t, w_M, b\rangle$. Here, A is an anti-knowledge base and D a collection of documents. Threshold t lower-bounds expected matching precision (thereby restricting the selection of matching methods) while w_M assigns weights to matches. Finally, b is a cost budget limiting computational overheads. The Detection Result is a set M of matches, ideally maximizing $\sum_{m \in M} w_M(m)$ under cost budget b while respecting precision threshold t.

3 System Overview

Figure 1 shows a high-level overview of the WebChecker system. Next, we describe its context and components quickly. In the following sections, we discuss specific components in more detail.

WebChecker accesses two large repositories: an anti-knowledge base (AKB), containing known misinformation, and a large document collection. WebChecker searches for text snippets in the document collection that match entries in the AKB. More precisely, it processes detection tasks, specified by a user. A detection task is characterized by a matching precision threshold, a function assigning weights to matches, and a cost budget (see Definition 5 for further details). The output is a set of matches, linking text snippets to AKB entries.

The goal of WebChecker is to find a set of matches that maximize accumulated weight, while staying within the cost budget. Furthermore, the result matches must be verified by a verification method that complies with the precision constraints. WebChecker uses several techniques to find high-quality matches efficiently.

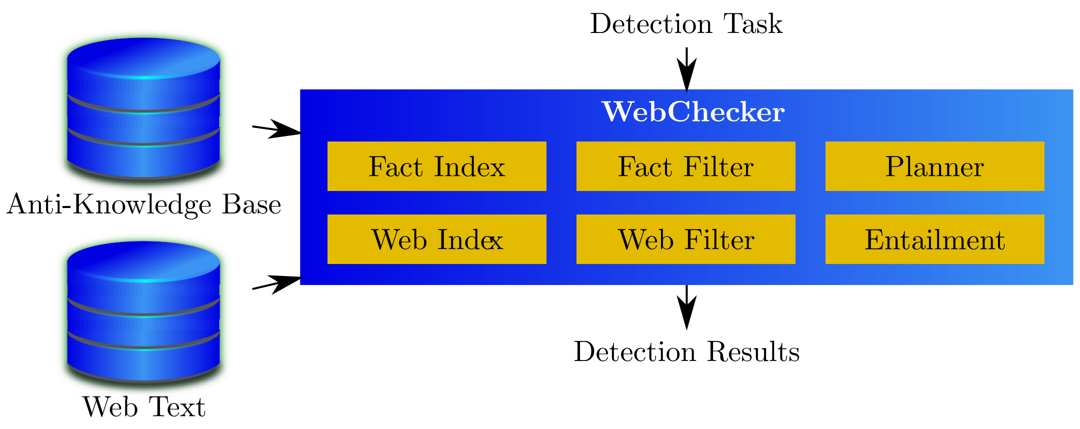

Figure 1: Overview of WebChecker system architecture.

First, WebChecker uses indexes to access subsets of anti-facts and documents efficiently. AKB entries are associated with tags, indicating for instance the source from which AKB entries have been extracted or the type of claim (e.g., distinguishing claims about numerical properties from claims about entity-to-entity relationships [24]). AKB entries are indexed, making it efficient to retrieve all entries associated with specific tags. On the other side, we assume that documents are indexed by keywords (e.g., by a Web search engine). This allows WebChecker to efficiently retrieve documents likely to contain matches for specific anti-facts. Second, WebChecker may use heuristic filters to narrow down the set of anti-facts and text snippets. Doing so is beneficial if it reduces the number of (more expensive) follow-up operations.

Ultimately, all result matches must satisfy an entailment check. Here, WebChecker verifies via natural language analysis whether specific text snippets entail AKB entries. If so, the corresponding snippet contains misinformation. Checking entailment via large neural networks (e.g., via Roberta [30]) is computationally expensive. Hence, WebChecker may use cheap but less precise checks to narrow down the focus for expensive checks. For instance, those checks may include simple checks for word overlap between AKB entry and text snippet. Alternatively, they may include checks via size-reduced language models that trade accuracy for overheads (e.g., via approaches such as Albert [26] and Distilbert [34]).

WebChecker has various options to process a given detection task. For instance, WebChecker can select the anti-facts for which to search matches via Web search (the optimal choice depends on the anti-fact weight as well as on the prevalence). Also, Web Checker may decide whether or not to activate heuristic filters and which sequence of entailment checks to perform. To make theses and other optimization choices, WebChecker uses a planning component.

Optimal planning decisions depend on factors that are unknown a-priori. For instance, it is difficult to estimate the prevalence of specific anti-facts. WebChecker does not assume that such statistics are available before processing starts. Instead, it learns to make near-optimal planning decisions via trial and error. The WebChecker planner uses an iterative algorithm. In each iteration, it uses a fixed percentage of the processing budget to find a maximal number of AKB matches. The accumulated weight of all matches retrieved in a single iteration serves as reward function in a reinforcement learning framework [40]. Thereby, existing algorithms from the reinforcement learning domain can be used to converge to optimal planning decisions (associated with maximal reward). Planning is described in more detail in Section 5.

4 Detection Plans

WebChecker processes detection plans to identify misinformation. We discuss their structure next.

Our goal is to match anti-facts in an anti-knowledge base A to text snippets in a document collection D. We assume an entailment check e is available that checks if a given text snippet entails a given anti-fact (if so, the text snippet contains misinformation). In its simplest form, a detection plan may be executed as a join between anti-facts and snippets, using the entailment check as join condition, i.e.

$$A \bowtie_e D.$$

The type of entailment check is restricted by the user-specified precision threshold t. For a low-precision threshold, relatively cheap entailment checks (e.g., via distilled language models such as DistilBERT [34]) can be used. Also, instead of a single entailment check, we can use a sequence of more and more expensive entailment checks, reserving expensive checks to matches that pass the cheaper tests. In doing so, we may reduce recall (due to false negatives of cheaper filters) but not precision (as long as the final filter satisfies the precision threshold). If cheap filters are sufficiently selective, overall efficiency increases.

We can use indexes on anti-facts as well as text documents to speed up matching. We consider scenarios where the document collection to analyze is extremely large. It is not realistic to create specific indexes for WebChecker on the entire Web. Fortunately, large parts of the Web have been indexed for keyword search by search engine providers such as Google or Microsoft. We assume that such an index is used to retrieve relevant documents for a given anti-fact. We denote the function that maps an anti-fact a to a keyword query as $i_D(a)$. For instance, we can form this keyword query by concatenating subject, predicate, and object keywords from the triple associated with a. Alternatively, we can use keywords from only a subset of those three components. WebChecker may create one or multiple indexes on anti-facts, indexing them for instance by mining source or topic. By i_A, we denote a condition on anti-facts that can be evaluated using an index on anti-facts. For instance, we can retrieve anti-facts from a specific source or of a specific type. In summary, plans with index usage can be described by the following expression:

$$\sigma_{i_A}(A) \bowtie_e \sigma_{i_D(a)}(D).$$

They are evaluated by retrieving single anti-facts satisfying condition i_A, then retrieving matching documents from D for each anti-fact a.

Finally, we can exploit cheap heuristics to filter out anti-facts and text snippets. We are interested in filtering out anti-facts that are unlikely to generate matches. Also, we want to filter text snippets that are unlikely to contain misinformation. We denote by h_A a heuristic filter on anti-facts and by h_D a heuristic filter on text snippets. For instance, we can exploit previously proposed methods for identifying check-worthy claims [15] to filter text snippets. For filtering anti-facts, we can exploit insights on which anti-fact properties make them likely to propagate (e.g., we show in prior work that mistakes are more likely for certain types of entities and predicates [24]). In summary, plans using heuristics and indexes can be written as

$$\sigma_{h_A}(\sigma_{i_A}(A)) \bowtie_e \sigma_{h_D}(\sigma_{i_D(a)}(D)).$$

WebChecker evaluates such plans by first using the index on A to retrieve anti-facts satisfying i_A and then filtering anti-facts via h_A. For each resulting anti-fact a, WebChecker uses the document index (i.e., the search engine) to retrieve documents satisfying i_D (up to a maximal number of documents) before filtering associated text snippets using heuristic h_D. Finally, the entailment checks e are executed between a and each remaining text snippet. Each snippet that entails a is added to the set of result matches.

5 Adaptive Planning and Processing

We discussed the general structure of detection plans in the previous section. That structure leaves open several optimization decisions. First, we can choose which filters (if any) to use on anti-facts and text snippets (heuristic filters and index-based filters). We assume that a set of alternatives is available for each type of filter (we discussed

Algorithm 3: Adaptive processing algorithm used by WebChecker.

```
1 WebChecker($A, D, t, w_M, b$) begin
2     $M \leftarrow \emptyset$;                                   // Initialize AKB matches
3     while Cost budget $b$ not reached do
4         $p \leftarrow$ SelectPlan($A, D, t$);                   // Select plan via RL
5         $N \leftarrow$ EvalOnSample($p, t$);                    // Evaluate new plan
6         $r \leftarrow \sum_{n \in N} w_M(n)$;                   // Calculate reward value
7         UpdateStats($p, r$);                                    // Update RL statistics
8         $M \leftarrow M \cup N$;                                // Add new matches
9     return $M$;                                                 // Return detection result
```

some alternatives in Section 4). Second, we can choose the sequence of entailment checks. Of course, any filter (or additional entailment check) tends to decrease the total number of matches if processing all anti-facts and text snippets. However, it is not realistic to search the entire Web. Instead, our goal is to maximize the number of matches found until the cost budget runs out. Given the cost budget, filters can increase the number of matches found as they help us focus our efforts on the most relevant AKB entries and text snippets.

Following Definition 5, the best plan maximizes the accumulated weight of all matches found under the cost budget. To select optimal plans, we would need to know the cost and accuracy of filter operations and entailment checks, as well as the frequency at which certain types of misinformation appear on the Web. In particular, having the latter kind of information about a large and dynamic document collection such as the Web does not seem realistic. This motivates a more flexible processing strategy that adapts to information gained at run time. More precisely, WebChecker implements the adaptive processing strategy described as Algorithm 3.

Given an anti-knowledge base A, a document collection D, a precision threshold t, a match weighting function w_M, and a cost budget b as input, WebChecker iterates until the cost budget is depleted (our current implementation measures cost as run time while other metrics, such as monetary processing fees, could be used instead). In each iteration, WebChecker picks a detection plan p and uses it for a limited amount of processing (in our current implementation, we limit ourselves to one AKB entry per iteration and 30 seconds of Web search time). The Web matches resulting from processing are added to the result set.

In each iteration, WebChecker picks plans based on statistics collected while processing the current detection task. In doing so, it faces the so-called exploration-exploitation dilemma: we have a tension between selecting plans that have worked well so far and selecting plans about which little is known (e.g., since they have not been tried yet). Reinforcement learning is the classical framework used to resolve this kind of tension in a principled manner. Hence, we apply a reinforcement learning algorithm to select the next plan to try in each iteration. As reward function (quantifying the quality of a plan), we use the sum of weights over all matches found in a given iteration. The reinforcement learning algorithm will therefore converge towards plans that return more and higher weighting matches.

More formally, the environment for a reinforcement learning algorithm is typically described as a Markov Decision Process (MDP). An MDP is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ of states $\mathcal{S}$, actions $\mathcal{A}$, transitions $\mathcal{T}$, and reward function $\mathcal{R}$. In our case, states correspond to vectors $\langle v_1, \ldots, v_n \rangle \in \mathbb{N}^n$ where each component $v_1, ..., v_{n-1}$ represents a plan property and v_n represents the plan property to decide next. Actions represent alternative values for the next plan property. The transition function maps a state $\langle v_1, \ldots, v_i, \ldots, v_{n-1}, i \rangle$ and an action a to the state $\langle v_1, \ldots, a, \ldots v_{n-1}, i + 1 \rangle$ (i.e., we set the value specified by the action and advance to the next plan property). After determining the last plan property, the current episode ends and the specified plan is evaluated. The reward for any transition is zero except for the last transition in an episode. For the last transition, the reward corresponds to the sum of weights over all matches collected by the specified plan in the

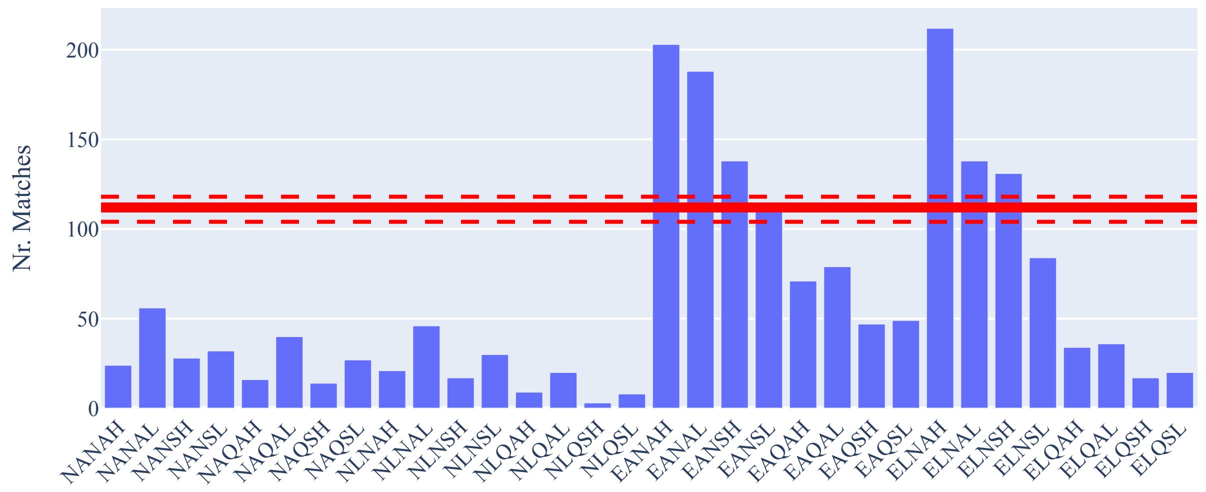

Figure 2: Number of misinformation matches found on the Web during one hour of processing time for 32 different plans. The red lines mark the number of matches found when optimizing the plan via reinforcement learning in three consecutive runs (solid line marks median).

current iteration. The transition function is deterministic while the reward function is stochastic (since using the same detection plan on different AKB entries in different iterations may yield a different number of matches).

6 Experimental Results

We report first experimental results for an early prototype of WebChecker. The prototype is implemented in Python 3. The source code is publicly available (`https://github.com/itrummer/WebChecker`).

The prototype uses an AKB consisting of over 100,000 entries (publicly available online at `https://github.com/cornelldbgroup/AntiKnowledgeBase`). Those entries were mined from Wikipedia update logs using the process described by Karagiannis et al. [24]. For querying the Web, the prototype uses Google's programmable search engine API [12]. For natural language processing, it uses Transformer models via the Huggingface Transformers library [47]. The prototype uses Stable Baselines 3 [39] for reinforcement learning. The experiments were executed on a p3.2xlarge EC2 instance with Tesla V100 GPU.

The prototype considers the following plan space properties. First, AKB entries are indexed by their type, distinguishing mistakes with incorrect numbers (e.g., incorrect birth dates) from mistakes with incorrect entities (e.g., an incorrect location for the headquarters of a company). The prototype may choose between those two categories to maximize detection yield. Second, the prototype may choose to filter AKB entries to the ones with long text (the intuition being that Web search results tend to be more specific and relevant for such triples). Third, the prototype may choose how to construct Web queries for a given AKB triple. In the current version, we consider two simple variants (either using quotes around triple subject and triple object or no quotes at all). Fourth, the WebChecker prototype may choose to filter Web results to Web sites with relatively small text content. Intuitively, more small Web sites can be processed for a fixed time limit, reducing the risk of investing significant computational resources into a Web site that turns out to be irrelevant. Finally, the prototype considers two different sequences of entailment checks. The first version directly applies a large Roberta model [30], pre-trained on MNLI (`https://huggingface.co/roberta-large-mnli`), to check whether a Web sentence entails the concatenation of components of an AKB triple. The second version performs a cheaper check first (testing for a sufficient ratio of overlapping words between triple and sentence). It applies the model-based test only if the cheaper one succeeds.

AKB Entry (with Correction)	**Web Match (with URL)**
MCS6501 microprocessors, was in, July 24 1974 issue of Electronics magazine **[Correct: July 24 1975]**	One of the earliest was a full-page story on the MCS6501 and MCS6502 microprocessors in the July 24, 1974 issue of Electronics magazine [`https://en-academic.com/dic.nsf/enwiki/12304`]
Biraj Bahu, is, 1955 Hindi film **[Correct: 1954]**	... and Biraj Bahu (1955), which won her the Filmfare Best Actress Award in 1955 [`https://en.wikipedia.org/wiki/Kamini_Kaushal`]
Haflioi Hallgrimsson, currently lives in, Edinburgh **[Correct: Bath]**	Haflioi Hallgrimsson, an Icelandic composer, currently living in Edinburgh [`https://en.wikipedia.org/wiki/Icelandic_diaspora`]
Acorda Therapeutics, is, biotechnology company based in Hawthorne **[Correct: Ardsley]**	Acorda Therapeutics (NASDAQ: ACOR, FWB: CDG) is a biotechnology company based in Hawthorne, New York [`https://www.morebooks.de/store/fr/book/acorda-therapeutics/isbn/978-620-1-18842-6`]

Table 14: Examples of Web matches found by WebChecker (all links were verified on 6/10/2021).

Considering binary choices for all five plan properties, the prototype considers a plan space of size 32 in total. In a first test, we ran all detection plans with a timeout of one hour per plan (we set a timeout of 30 seconds for finding matches for a single AKB entry and restrict the number of considered AKB entries to 120). The goal was to verify that planning choices have significant impact on the number of matches found. In a second test, we use the AC2 [31] reinforcement learning algorithm to make planning choices automatically (using the number of Web matches found as reward function). Note that we use uniform weights for those initial experiments (i.e., every Web match counts equally). Also, we do not explicitly consider precision thresholds (all possible plans use the same, final entailment check via the Roberta Transformer). Adding more fine-grained precision control is one of the future work avenues discussed in Section 8.

Figure 2 shows the results of both experiments. The y axis represents the number of Web matches found within one hour. The x axis describes plans in short notation (each letter describes one plan property in the following order): N for number mistakes versus E for entity mistakes, A for all triples versus L for long triples, N for non-quoted versus Q for quoted Web queries, A for all versus S for short Web sites, and H for (direct) high precision entailment checks versus L for an initial low-precision check. The solid red line represents the median number of matches found in three consecutive runs when using reinforcement learning for planning (dashed lines represent the number of matches for the other two runs).

The number of matches found by different plans differs by two orders of magnitude (ranging from three matches for plan NLQSH to 212 for plan ELNAH). For instance, AKB entries indexed as entity-related mistakes tend to produce more Web matches, compared to number-related mistakes (24 versus 98 matches when averaging over all plans). As another example, leaving out quotes in Web queries tends to increase recall significantly (91 versus 31 matches in average over all plans). Note, however, that different planning choices are dependent on each other and cannot be easily separated (e.g., enabling low-precision checks increases recall in many cases but not in all). This motivates the use of sophisticated planning approaches.

Reinforcement learning consistently finds plans with a quality (i.e., number of matches) significantly above average. In all three runs, the plans produced via reinforcement learning were better than 25 out of 32 plans with a median of 112 matches (compared to a median of 35 matches over all plans). This shows that reinforcement learning is suitable for finding good plans within the detection plan space.

Table 14 shows example matches found by WebChecker. Clearly, matches found cover a variety of topics (reflecting the diversity of content of the AKB we use). The current prototype uses the same final entailment check for all candidate matches. Nevertheless, planning choices may influence not only recall but also precision of the final result. We hand-verified entailment for a sample of 20 matches from the plan generating most matches (ELNAH) and hand-verified all matches for the plan generating the least matches (NLQSH). We found a precision of 100% for the latter but only a precision of 24% for the former. While our sample is small, this indicates that more work is needed to optimize precision during retrieval. We discuss first ideas in Section 8.

7 Related Work

Our work connects to a large body of recent work that exploits machine learning for automated fact-checking [6, 18, 22, 27, 29], including work focused on the platforms we cite as motivating use cases [19, 35]. In particular, our research connects to prior work using language models for fact checking [6, 27]. We refer to recent surveys for a detailed overview of corresponding approaches [4, 38]. What distinguishes our work from prior contributions is our focus on computational efficiency, exploiting ideas from the database community to scale up automated fact checking.

We use Web search for automated fact checking. That connects our work to prior checking methods based on Web search engines [8, 11, 45]. However, prior work typically uses Web search to retrieve relevant documents for verifying a given claim. Instead, we use it to retrieve instances of misinformation from the Web. In that, our work is similar to the one by Elyashar et al. [8]. Here, the focus is on collecting fake news documents via Web search. However, our work differs as Web search is only one step in a multi-step verification pipeline. Our contribution is in an infrastructure that takes planning decisions maximizing misinformation retrieval efficiency.

Multiple branches of prior work could be used to extend WebChecker. For instance, a popular branch of fact checking research focuses on identifying check-worthy claims [13–15, 33]. If sufficiently efficient, such methods can be used as filters on Web results before applying more expensive verification methods. Also, we currently identify misinformation by comparing against entries in an AKB [1, 2, 24]. In future work, we may consider other verification sources such as knowledge graphs [29] or relational databases [13, 20, 22, 23].

Finally, we use techniques from the database area to make large-scale fact checking efficient. In particular, we exploit adaptive processing and query planning techniques [3, 41, 42]. Reinforcement learning has been used for optimizing adaptive query plans before [42]. However, our plan space, processing engine, and reward function are specific to the domain of fact checking. Our goal is not to process all input data for a given query. Instead, it is to produce the maximal number of matches until a time budget runs out. More broadly, our work is part of a research direction that transfers techniques from the database community to new use cases (e.g., recent work exploiting declarative query optimization to make visual data processing more efficient [21, 25]). In this case, we transfer ideas such as query planning to the domain of automated fact checking.

8 Conclusion and Outlook

Our goal is to make large-scale, automated fact checking efficient. We present a first prototype of WebChecker, a system that searches for misinformation in a large document collections while minimizing computational overheads. To do so, WebChecker exploits a diverse set of operators and an adaptive optimizer to choose between them.

We observe the following in our experiments. First, planning choices can significantly impact efficiency in automated fact checking. Comparing best and worst plans, we find a two orders of magnitude difference in computational overheads per (misinformation) detection. Second, we find that adaptive processing and reinforcement learning are effective at identifying good detection plans. While less efficient than the optimum by a factor of about two (rate of detection), learned plans outperform the majority of plans in the plan space.

Our current prototype does not yet support user-defined precision constraints (we outline in Section 2 why such constraints are required to adapt WebChecker to different scenarios). Also, our experiments indicate that output precision depends on various plan properties (beyond the final entailment check). In a first approach, we plan to consider operator-specific accuracy statistics (gained, for instance, via small-scale experiments on pieces of misinformation for which manually generated ground truth is available) during planning. Alternatively, we plan to explore approaches that use feedback by crowd workers on result samples to evaluate plan precision. Beyond precision, we plan to increase computational efficiency by expanding WebChecker's set of operators (e.g., by adding entailment checks by smaller models [26,34] as optional filters). Also, we plan to study parallel processing as a means to increase verification throughput.

Acknowledgement

This project is supported by a Google Faculty Research Award ("Mining an Anti-Knowledge Base for Fact Checking from Wikipedia Updates"). We thank the editors for helpful suggestions and comments on an earlier version of this paper.

References

[1] H. Arnaout, S. Razniewski, and G. Weikum. Negative Statements Considered Useful. 2020.

[2] H. Arnaout, S. Razniewski, G. Weikum, and J. Z. Pan. Negative Knowledge for Open-world Wikidata. *Wikimedia Workshop*, 2, 2021.

[3] R. Avnur and J. Hellerstein. Eddies: continuously adaptive query processing. In *SIGMOD*, pages 261–272, 2000.

[4] A. Bondielli and F. Marcelloni. A survey on fake news and rumour detection techniques. *Information Sciences*, 497:38–55, 2019.

[5] B. Byrne, K. Krishnamoorthi, S. Ganesh, and M. S. Kale. TicketTalk: Toward human-level performance with end-to-end, transaction-based dialog systems. 2020.

[6] G. Demartini, S. Mizzaro, and D. Spina. Human-in-the-loop Artificial Intelligence for Fighting Online Misinformation: Challenges and Opportunities. (September):65–74, 2020.

[7] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, volume 1, pages 4171–4186, 2019.

[8] A. Elyashar, M. Reuben, and R. Puzis. Fake News Data Collection and Classification: Iterative Query Selection for Opaque Search Engines with Pseudo Relevance Feedback. 2020.

[9] L. Floridi and M. Chiriatti. GPT-3: Its Nature, Scope, Limits, and Consequences. *Minds and Machines*, 30(4):681–694, 2020.

[10] Fullfact. The state of automated factchecking. Technical report, 2016.

[11] D. Gerber, D. Esteves, J. Lehmann, L. Bühmann, R. Usbeck, A. C. Ngonga Ngomo, and R. Speck. DeFacto - Temporal and multilingual deep fact validation. *Journal of Web Semantics*, 35:85–101, 2015.

[12] Google. https://programmablesearchengine.google.com/about/, 2021.

[13] N. Hassan, F. Arslan, C. Li, and M. Tremayne. Toward automated fact-checking: detecting check-worthy factual claims by ClaimBuster. In *SIGKDD*, pages 1803–1812, 2017.

[14] N. Hassan, C. Li, and M. Tremayne. Detecting check-worthy factual claims in presidential debates. *International Conference on Information and Knowledge Management, Proceedings*, 19-23-Oct-2015:1835–1838, 2015.

[15] N. Hassan, G. Zhang, F. Arslan, J. Caraballo, D. Jimenez, S. Gawsane, S. Hasan, M. Joseph, A. Kulkarni, A. K. Nayak, V. Sable, C. Li, and M. Tremayne. ClaimBuster: the first-ever end-to-end fact-checking system. *VLDB*, 10(7):1–4, 2017.

[16] J. Howard and S. Ruder. Universal Language Model Fine-tuning for Text Classification. In *ACL*, pages 328–339, 2018.

[17] T. Hughes, J. Smith, and A. Leavitt. Helping People Better Assess the Stories They See in News Feed with the Context Button. *Facebook*, 2018.

[18] K. Hunt, P. Agarwal, and J. Zhuang. Monitoring Misinformation on Twitter During Crisis Events: A Machine Learning Approach. *Risk Analysis*, 00(00), 2020.

[19] S. Jain, V. Sharma, and R. Kaushal. Towards automated real-time detection of misinformation on Twitter. *2016 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2016*, pages 2015–2020, 2016.

[20] S. Jo, I. Trummer, W. Yu, X. Wang, C. Yu, D. Liy, and N. Mehta. AggChecker: a fact-checking system for text summaries of relational data sets. *VLDB*, 12(12):1938–1941, 2019.

[21] D. Kang, P. Bailis, and M. Zaharia. Blazelt: Optimizing declarative aggregation and limit queries for neural networkbased video analytics. *VLDB*, 13(4):533–546, 2019.

[22] G. Karagiannis, M. Saeed, P. Papotti, and I. Trummer. Scrutinizer: A mixed-initiative approach to large-scale, data-driven claim verification. *VLDB (Conditionally Accepted - Preprint: https://arxiv.org/abs/2003.06708)*, 2020.

[23] G. Karagiannis, M. Saeed, P. Papotti, and I. Trummer. Scrutinizer: a mixed-initiative approach to large-scale, data-driven claim verification [Extended Technical Report]. Technical report, 2020.

[24] G. Karagiannis, I. Trummer, S. Jo, S. Khandelwal, X. Wang, and C. Yu. Mining an "anti-knowledge base" from Wikipedia updates with applications to fact checking and beyond. *VLDB*, 13(4):561–573, 2020.

[25] S. Krishnan, A. Dziedzic, and A. J. Elmore. DeepLens: Towards a visual data management system. *CIDR 2019 - 9th Biennial Conference on Innovative Data Systems Research*, 2019.

[26] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. pages 1–17, 2019.

[27] N. Lee, B. Li, S. Wang, W.-t. Yih, H. Ma, and M. Khabsa. Language Models as Fact Checkers? pages 36–41, 2020.

[28] A. H. Li and A. Sethy. Knowledge Enhanced Attention for Robust Natural Language Inference. 2019.

[29] J. Liu, C. Wang, C. Li, N. Li, J. Deng, and Z. Pan. DTN: Deep Triple Network for Topic Specific Fake News Detection. In *Web Semantics: Science, Services and Agents on the World Wide Web*, page 100646. Elsevier B.V., 2021.

[30] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv*, (1), 2019.

[31] V. Mnih, A. P. Badia, L. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *33rd International Conference on Machine Learning, ICML 2016*, 4:2850–2869, 2016.

[32] E. O. Nsoesie and O. Oladeji. Identifying patterns to prevent the spread of misinformation during epidemics. *Harvard Kennedy School Misinformation Review*, 2020.

[33] S. Rosenthal and K. McKeown. Detecting opinionated claims in online discussions. *Proceedings - IEEE 6th International Conference on Semantic Computing, ICSC 2012*, pages 30–37, 2012.

[34] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. pages 2–6, 2019.

[35] A. Sathe, S. Ather, T. M. Le, N. Perry, and J. Park. Automated fact-checking of claims from wikipedia. *LREC 2020 - 12th International Conference on Language Resources and Evaluation, Conference Proceedings*, (May):6874–6882, 2020.

[36] M. Scott. 'It's overwhelming': On the frontline to combat coronavirus 'fake news'. *Politico*, pages 1–16, 2020.

[37] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *SIGMOD*, pages 23–34, 1979.

[38] K. Shu, A. Sliva, S. Wang, J. Tang, and H. Liu. Fake News Detection on Social Media: A Data Mining Perspective. *SIGKDD Explorations*, 19(1):22–36, 2017.

[39] Stable Baselines. https://github.com/DLR-RM/stable-baselines3.

[40] R. R. S. Sutton and A. G. A. Barto. *Reinforcement learning: an introduction*. 1998.

[41] I. Trummer, S. J. Mosley, J. Antonakakis, and S. Jo. SkinnerDB : regret-bounded query evaluation via reinforcement learning. *VLDBJ*, 11(12):2074–2077, 2018.

[42] I. Trummer, J. Wang, D. Maram, S. Moseley, S. Jo, and J. Antonakakis. SkinnerDB: regret-bounded query evaluation via reinforcement learning. In *SIGMOD*, pages 1039–1050, 2019.

[43] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017-Decem(Nips):5999–6009, 2017.

[44] K. Vault, H. Rank, and S. By. Google ' s Knowledge Vault Helps Rank Sites By Accuracy. *Popular Science*, pages 1–4, 2015.

[45] X. Wang, C. Yu, S. Baumgartner, and F. Korn. Relevant Document Discovery for Fact-Checking Articles. In *WWW*, pages 525–533, 2018.

[46] C. Wardle and E. Singerman. Too little, too late: Social media companies' failure to tackle vaccine misinformation poses a real threat. *The BMJ*, 372:1–3, 2021.

[47] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush. Transformers: State-of-the-Art Natural Language Processing. pages 38–45, 2020.