Future Generation Computer Systems 125 (2021) 232–246

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Detecting malicious behavior in social platforms via hybrid knowledge- and data-driven systems

Jose N. Paredes [a,1], Gerardo I. Simari [a,*,1,2], Maria Vanina Martinez [b,3], Marcelo A. Falappa [a,1]

[a] San Andrés 800, Campus Palihue, (8000) Bahía Blanca, Argentina
[b] Pabellón I, Ciudad Universitaria, (C1428EGA) Buenos Aires, Argentina

ARTICLE INFO

Article history:
Received 10 January 2021
Received in revised form 18 May 2021
Accepted 19 June 2021
Available online 25 June 2021

Keywords:
Malicious behavior
Fake news
Botnets
Social data
Information/misinformation diffusion
Decision support systems
Human-in-the-loop computing

ABSTRACT

Among the wide variety of malicious behavior commonly observed in modern social platforms, one of the most notorious is the diffusion of fake news, given its potential to influence the opinions of millions of people who can be voters, consumers, or simply citizens going about their daily lives. In this paper, we implement and carry out an empirical evaluation of a version of the recently-proposed NetDER architecture for hybrid AI decision-support systems with the capability of leveraging the availability of machine learning modules, logical reasoning about unknown objects, and forecasts based on diffusion processes. NetDER is a general architecture for reasoning about different kinds of malicious behavior such as dissemination of fake news, hate speech, and malware, detection of botnet operations, prevention of cyber attacks including those targeting software products or blockchain transactions, among others. Here, we focus on the case of fake news dissemination on social platforms by three different kinds of users: non-malicious, malicious, and botnet members. In particular, we focus on three tasks: (i) determining who is responsible for posting a fake news article, (ii) detecting malicious users, and (iii) detecting which users belong to a botnet designed to disseminate fake news. Given the difficulty of obtaining adequate data with ground truth, we also develop a testbed that combines real-world fake news datasets with synthetically generated networks of users and fully-detailed traces of their behavior throughout a series of time points. We designed our testbed to be customizable for different problem sizes and settings, and make its code publicly available to be used in similar evaluation efforts. Finally, we report on the results of a thorough experimental evaluation of three variants of our model and six environmental settings over the three tasks. Our results clearly show the effects that the quality of knowledge engineering tasks, the quality of the underlying machine learning classifier used to detect fake news, and the specific environmental conditions have on smart policing efforts in social platforms.

1. Introduction

One of the biggest problems that our overly-communicated society is suffering is information disorder, which can lead to misinformation and polarization of opinions and ideas, and even promote violent behavior. The diffusion of fake news on the Web has the potential to influence the opinions of millions of people who can be voters, consumers, or simply citizens going about their daily lives. Many initiatives are being carried out with the goal of fighting such phenomena, ranging from the deployment of policies and protocols to prevent the creation of such content, to technological research and development for identifying and tracking fake news. Since we believe that in this effort it is crucial to understand the complex socio-technological ecosystem of information diffusion, in this work we propose the implementation and evaluation of a decision-support system with the capability of reasoning about unknown information related to objects and actors, and the diffusion processes that affect them.

Our proposal is based on the NetDER architecture [1], which takes its name from "*Network Diffusion and Existential Rules*". The main idea behind this model is to afford the creation of hybrid artificial intelligence models that combine the use of data analytics and machine learning-based tools, ontological knowledge, and diffusion processes, leveraging the advantages from

* Corresponding author.
E-mail addresses: jose.paredes@cs.uns.edu.ar (J.N. Paredes), gis@cs.uns.edu.ar (G.I. Simari), mvmartinez@dc.uba.ar (M.V. Martinez), mfalappa@cs.uns.edu.ar (M.A. Falappa).
[1] Department of Computer Science and Engineering, Universidad Nacional del Sur (UNS) and Institute for Computer Science and Engineering (UNS–CONICET).
[2] School of Computing, Informatics, and Decision Systems Engineering (CIDSE), Arizona State University, Tempe, AZ 85281, USA.
[3] Department of Computer Science, Universidad de Buenos Aires (UBA) and Institute for Computer Science Research (UBA–CONICET).

https://doi.org/10.1016/j.future.2021.06.033
0167-739X/

reasoning in complex and explainable ways while having the ability to represent, forecast, and make sense of complex diffusion processes in social networks. The NETDER architecture is designed to tackle several suspicious/malicious behavior problems, which sometimes can be related or may arise in completely different contexts. For example, on the one hand, NETDER could be instantiated to solve two specific related problems such as detecting fake news posts on social media and determining if social bots are used for this purpose (works such as [2] suggest the relationship between these problems; we present more details about such issues in Section 3). On the other hand, in a completely different context, NETDER could also be used to detect malicious behavior in blockchain transactions—in this domain, efforts have been carried out that use machine learning algorithms trained with graph properties obtained from the Ethereum blockchain [3]; in [4], Bitcoin users are labeled in a semi-automatic way based on information regarding their identity and actions, which is automatically scraped. Such works show that it is possible to identify suspicious or malicious behavior patterns in blockchain transactions; our proposed framework would allow the combination of such data-intensive processing with expert domain knowledge and perhaps other pertinent behavior information from users in Darknet forums or other social platform sources that can help improve the resulting model and make it more robust and explainable.

As we will see in Section 5, most works focused on solving this kind of problems are based on *ad hoc* methods, and address only specific variants of malicious behavior; NETDER, on the other hand, is designed to develop more general tools that can potentially combine different approaches to solve a wide variety of related problems. The model, as shown in Fig. 1, involves a data ingestion module that handles issues such as data cleaning, schema matching, inconsistency, incompleteness, data analytics, etc., as well as other higher-level issues such as trust and uncertainty management; this module is fed by a variety of data sources (assumed to be updating at independent rates). The architecture also has the following two main modules:

- *Ontological Reasoning Module* (ORM): Stores the knowledge base (comprised of facts, rules, and constraints) both for background (domain) knowledge as well as for the network. It provides inference services for query answering; the network diffusion module plays a supporting role, offering network queries as a service to this module.
- *Network Diffusion Module* (NDM): Models the dynamic aspects of networks in the form of diffusion processes, computing inferences based on the network knowledge base stored in the ORM. The main services implemented here are therefore the simulation of such processes and checking conditions over the network for the ORM, which can be leveraged in forecasting if certain situations of interest will occur.

Finally, the *Query Answering Module* (QAM) is in charge of implementing the main reasoning task, which is called NETDER Query Answering Process. This module is in charge of coordinating the execution of both ORM and NDM modules in order to produce results (*i.e.*, answers to specific queries that users issue to the system).

In the next section, we will discuss a simple use case illustrating the application of this general architecture in a social media setting. But first, we briefly focus on a simple illustration of how the NDM can be used in this same setting to help the reader to grasp the intuition behind this framework; Fig. 2 shows how diffusion models are leveraged to forecast the spread of exposure to malicious content throughout a social network.[4] The example shows how two different diffusion processes, starting both at the same state (the situation at the current time point) model the future spread of such content after two time steps, arriving at different estimates of the percentage of users that will be exposed to the malicious content. As we will see later on, having a diffusion model that is capable of accurately forecasting how the world will actually evolve is paramount to achieving high levels of precision and recall in the main query answering tasks that the tool is designed to carry out. Nevertheless, a tool such as NETDER can be used to simulate different diffusion hypotheses and use the results in defining effective de-risking and contingency policies for live scenarios; the events surrounding the 2020 US presidential election and its lasting effects leading to political unrest are a clear example of the usefulness that these kinds of tools may have if used adequately.

A simple use case

To understand in more detail the roles and components of each module, consider the following example, which is a simplification of the setting developed for the experimental analyses in Sections 3 and 4.

Suppose we have a social platform where a set of users share information by posting news articles (creating new posts or reposting existing ones). The NDM in this case will have a formal representation of the social platform in the form of a complex graph that includes a set of nodes and edges, and a diffusion process to simulate how the news articles flow among the users. The nodes and edges can have several labels, each with a different *weight*, representing uncertain knowledge such as the likelihood or confidence that a certain property holds. There are also *global* labels to represent the general state of the network—their values therefore depend on those of the local labels. Naturally, confidence in the knowledge represented by these labels is dynamic, and one way to capture this dynamism is using logical rules.

In our example, each node (user) has associated a label that corresponds to that user's "preferred news category", which could be something like sports, culture, politics, etc. The users' preferences about news topics may be dynamic, and change as posts circulate in the network. A simple rule in the diffusion model to update the *preferred category* label of a user (whose preferences are affected by what they see in their feeds) could state that for each user u at current time t:

> **Diffusion Rule:** "*if the majority of u's neighbors, who have published at time t, have* **C** *as their preferred category, then the preferred category for u at time* $t+1$ *will be* **C** *as well*".

For example, if u's neighbors are u_1 and u_2, who both posted about "sports" (her preferred category) at time t, and u_3 who posted about "politics" (her preferred category) at time t, then this diffusion rule predicts that u will prefer to post about "sports" at time $t+1$. Note that time is represented explicitly, and the values of labels depend on the specific time points or intervals considered.

The whole diffusion process is defined by a set of rules like the one we mentioned above that update information in nodes and edges. One can check the state of the labels in the network by means of a process that applies the rules in a saturated manner (until no new information is added, or a maximum number of time points – or horizon – is reached). Returning to Fig. 2, we can see that the malicious content spreads faster in β than in α,

[4] Intuitively, one can also picture the animations commonly seen in weather forecasts to understand the usefulness of this feature.

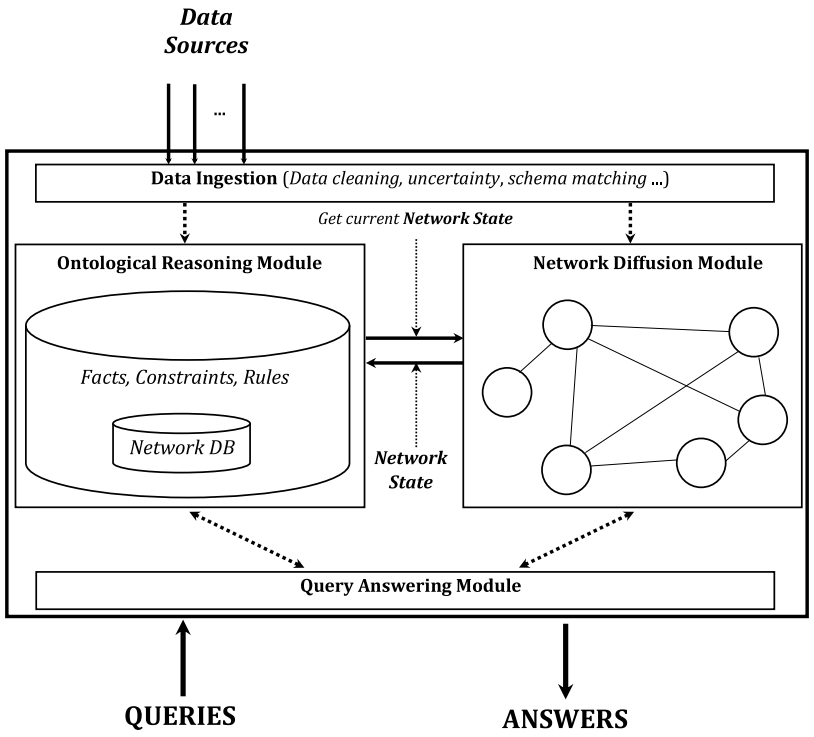

Fig. 1. Architecture sketch for the NETDER framework [1].

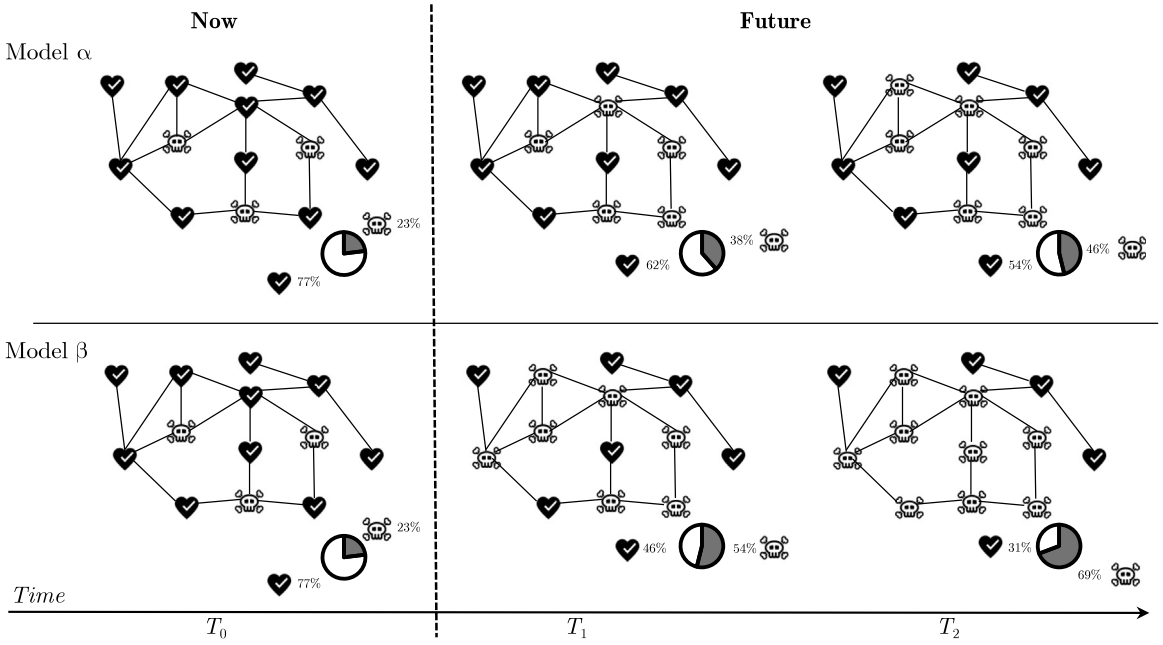

Fig. 2. Illustration of the evolution of a network's state using two different diffusion models: α (top) and β (bottom); users that have been exposed to malicious content (such as fake news) are represented with skulls, while the rest are depicted with hearts. The figure shows the forecasts made by the two models, starting at T_0 (now) and continuing for two time steps; α forecasts that 46% of users will be exposed after this time, while β is less optimistic.

though both cases actually depict simulations and the actual evolution will be closer to one or the other. Hence, as with well-known simulations such as weather forecasts, models that better fit with reality provide greater possibilities to curb user exposure to malicious content.

On the ORM side, the knowledge base will contain:

(i) The set of facts or statements that represent the structural information of the network—that is, statements such as *u and v are users*, *u and v are friends*, *the preferred category of u at time t is believed to be* **C**, etc.

(ii) A set of facts about the domain, which in this case could be information about the news articles, who posts what, the categories to which each news article belongs, if a news article is considered to be fake news and the confidence level with which that holds, etc. Some of these facts may be input from the ingestion module, which processes the raw information about objects of interest and performs statistics or more complex data

analytics over them, such as detecting the category of a news article, deciding if a news article could be considered to be fake news and with what confidence, or if a user can be considered to be an early poster of a specific news article.
(iii) A set of rules that allow to make inferences leveraging the information in (i) and (ii). For instance, suppose we are interested in identifying users that are responsible for the dissemination of fake news. A rule that would allow to flag such users might state:

> **Ontological Inference Rule:** "*if user u is found to be an early poster of an article that is believed to be fake news with high confidence, then it is possible that u is responsible for disseminating fake news*".

Such rules are expected to be defined by an expert in the domain with experience in detecting and analyzing malicious behavior. Alternatively, explicit policies such as those for *coordinated inauthentic behavior* defined by Facebook[5] can be used as baselines to define rules. It is also possible to learn rules from existing data and reports via data mining and machine learning tools. One of the advantages of deriving results using this kind of rules is that it naturally supports their *explainability*—as we discuss in Section 5, most tools for addressing malicious behavior in social platforms are based on the direct application of machine learning techniques, which tend to result in black boxes.

The framework allows for complex inferences that involve incomplete and dynamically changing knowledge. The reasoning mechanism that NetDER offers is performed by means of a *query answering* process, which is implemented by the Query Answering Module (cf. Fig. 1), that emerges from the collaboration of both modules—they inform each other, potentially generating new inferences as a consequence. In the case of our example, we could ask the system to provide the list of all users that are suspected of being responsible for disseminating fake news. In this case, the ORM will query the NDM asking for the current state of the specific labels of nodes and edges that can yield pertinent results. The network module will run its simulation (diffusion process) up to the horizon (maximum number of time points) and return the values for the labels, which may update the information maintained in the ORM. After this, the ORM will run its own inference process (this mechanism is designed in [1] as an adaptation of the well-known *chase* procedure used in rule-based ontologies [5]), firing applicable rules until an answer is obtained. This query answering mechanism is called *one-shot* in [1]; there are other possible interactions among the modules that would yield different semantics (and computational costs). Details on syntax and semantics for query answering in the NetDER framework can be found in [1] and [6]. In the following sections, we show how the different components of the architecture are instantiated with data and information to reason about fake news articles, malicious users, and botnet structures.

Contributions and Organization. The contributions of this work are the following: first, we develop the BadBot testbed that generates hybrid real-world/synthetic datasets with ground truth designed to feed evaluations of the performance of tools specifically designed to address tasks related to detection of malicious behavior in social platforms. Second, we discuss in detail how to instantiate all modules and components of NetDER so that the malicious behavior detection tasks are solved through a set of pertinent queries posed to the system. Third, using BadBot we create a series of scenarios that allow the evaluation of the performance of NetDER over these tasks. Finally, we report on the results of an extensive empirical evaluation of the derived models over these scenarios in order to show the practical applicability of the approach.

The BadBot testbed is thoroughly defined in Section 2, including the description of the particular tasks we are interested in addressing. Sections 3 and 4 – the core of the paper – describe how BadBot and NetDER are instantiated, discuss particulars of how performance evaluation is carried out, and present the results of an extensive empirical evaluation of three models in six settings. Finally, Sections 5 and 6 provide insights on related work and conclusions, respectively.

2. BadBot: A testbed for diffusion of malicious posts by single actors and botnets

We now present the details of a testbed[6] designed to generate datasets with all the information necessary to evaluate the performance of three different tasks centered around the detection of malicious behavior in social platforms:

1. Determine who is responsible for the initial posting of a malicious post; this task is referred to as Responsible.
2. Determine who is a malicious actor—in this setting, *malicious* refers to actors who purposefully disseminate malicious posts; this task is called Malicious.
3. Determine who is a member of a botnet designed to disseminate malicious posts—here, botnets are coordinated sets of nodes that post the same content (which can be either malicious or not) at the same time. This task is called Member.

Note that our general methodology can be adopted to create similar testbeds designed to evaluate other sets of tasks. This is, to the best of our knowledge, the first effort in creating a testbed for the evaluation of tasks for detecting more than one kind of malicious behavior at once, which is necessary for the evaluation of a decision support system like NetDER. We call our testbed "BadBot", short for *Bad actors and Bots*. We have the following basic components:

- *post-dataset*: A dataset of posts tagged with ground truth in order to distinguish genuine content from malicious one (for instance, real news from fake, or benign links and those related to malware). Each post has an assigned *category*; for instance, if posts are news articles they can be in category Sports, Arts, World, Politics, etc.
- $\mathcal{G} = (V, E)$: A graph of nodes and edges to represent a social network structure, where each node represents a user and each edge a relation (such as follow, friendship, fan, etc.). Each node in V has a flag stating whether or not it is malicious in nature. Furthermore, each node has associated a probability distribution of post categories, indicating its preferences.
- A series of *time points* starting at 0 and ending at t_{sim}. Depending on the instantiation of the testbed, each time point may represent different granularities of time, such as an hour, a day, a week, etc.—the values of the parameters in Fig. 3 should thus be adjusted accordingly.

Trace generation methodology. The testbed's main purpose is to generate full *traces* of actions occurring at different times, according to a set of parameters that characterize a desired setting, as described in Fig. 3. The process of generating traces is as follows (cf. Section 3 for an example of how it is implemented in the context of our experimental evaluation):

[5] https://www.facebook.com/communitystandards/inauthentic_behavior

[6] The source code for BadBot is made publicly available at https://github.com/jnparedes/BadBot.

Parameter	Description
new-graph	Flag indicating if a new graph should be created.
num_{nodes}	Number of nodes in graph $\mathcal{G}$.
num_{edges}	Number of edges in graph $\mathcal{G}$.
t_{sim}	Number of time points in the generated traces.
$prop_{mal}$	Proportion of malicious nodes in $\mathcal{G}$.
num_{botnet}	Number of botnets to be generated.
$prob_{memb}$	Probability that a malicious node is a botnet member.
$prob_{nm\_post}$	Probability that a non-malicious node posts an article (malicious or not) at each time point.
$prob_{m\_post}$	Probability that a malicious node posts an article (malicious or not) at each time point.
$prob_{b\_post}$	Probability that a botnet (i.e., all members) posts an article (malicious or not) at each time point.
$prob_{nm\_mal}$	Probability of choosing a malicious post when a non-malicious node creates a new post.
$prob_{m\_mal}$	Probability of choosing a malicious post when a malicious node creates a new post.
$prob_{b\_mal}$	Probability of choosing a malicious post when a botnet creates a new post.
$prob_{nm\_share}$	Probability that a non-malicious node shares a post from its neighbors.
$prob_{m\_share}$	Probability that a malicious node shares a post from its neighbors.
$prob_{b\_share}$	Probability that a botnet shares a post from its neighbors.

Fig. 3. Set of parameters of the BadBot testbed.

(1) Setup:
(a) If $new-graph = true$, create graph $\mathcal{G} = (V, E)$ with $|V| = num_{nodes}$ and $|E| = num_{edges}$.
(b) For each node in $|V|$, set its malicious flag according to $prop_{mal}$.
(c) For each malicious node in $|V|$, set its botnet flag according to $prob_{memb}$.
(d) Initialize *trace* as an empty data structure mapping from time points and nodes to posts.
(2) For $time_i$ ranging from 0 to t_{sim}:
(a) Each *non-malicious*, *malicious*, and *botnet* node decides, according to $prob_{nm\_post}$, $prob_{m\_post}$, and $prob_{b\_post}$, respectively, if they will create a new post or not.
If they do, they choose between malicious or benign according to $prob_{nm\_mal}$, $prob_{m\_mal}$, and $prob_{b\_mal}$, respectively; and then make a selection within *post-dataset* according to the outcome and their category preference.
(b) If a new post is not created, *non-malicious*, *malicious*, and *botnet* nodes decide, according to $prob_{nm\_share}$, $prob_{m\_share}$, and $prob_{b\_share}$, respectively, if they will share something posted by their connections, chosen among those having the same category as the *dominant* category of the posts made by neighbors.
The dominant category is determined by first calculating the most frequent category posted by each neighbor, and then taking the most frequent among those categories.
(c) Record in *trace* all the information created, including posts and their authors, made at $time_i$.
(3) Output *trace*.

In Section 3.1, we will describe how we instantiated the BadBot testbed in the general setting of diffusion of fake news.

3. Designing an experimental evaluation

In this section, we start by describing in detail how the BadBot testbed and NetDER architecture are instantiated to carry out our empirical evaluation (Sections 3.1 and 3.2), then discuss particulars of how performance evaluations are carried out (Section 3.3), before presenting the results in Section 4.

3.1. Basic setup: Instantiation of the BadBot testbed

The application domain is instantiated based on our testbed, choosing values for the different parameters and data with the goal of creating a realistic environment. The set of posts is comprised of news articles based on a selection of 10,000 items from [7]; since that dataset does not include categories, we randomly assign one out of five possible artificial categories to each article ($categ_1$, $categ_2$, $categ_3$, $categ_4$, or $categ_5$). In order to provide structure to the random assignment of categories to fake news articles, we choose $categ_1$ with probability 0.7, and the rest of the categories with probability $0.3/4 = 0.075$, while for non-fake news the assignment is done via a uniform probability distribution. This structure aims to represent, for instance, that fake news articles are more likely to be associated with a topic related to the opinion that is targeted for manipulation, such as a specific political party in the context of an election.

The social network graph $\mathcal{G} = (V, E)$ is randomly generated based on the R-MAT algorithm [8,9],[7] which is designed to create instances that imitate the structure that is observed in real-world data. For our evaluation, we set $num_{nodes} = 150$ and $num_{edges} = 495$ to build each instance, which we found to strike a good balance between size, density, and computational cost of performing many runs. For the length of each trace, we set $t_{sim} = 15$, which based on a 12-hour granularity corresponds to about a week of posting activity. Finally, other fixed parameters are set as follows:

num_{botnet}	$= 1$	$prob_{m\_mal}$	$= 0.6$
$prob_{nm\_post}$	$= 0.05$	$prob_{b\_mal}$	$= 0.6$
$prob_{m\_post}$	$= 0.5$	$prob_{nm\_share}$	$= 0.2$
$prob_{b\_post}$	$= 0.5$	$prob_{m\_share}$	$= 0$
$prob_{nm\_mal}$	$= 0.1$	$prob_{b\_share}$	$= 0$

These values were chosen with the following properties in mind, which reflect one possible setting that we considered to be well-balanced with respect to the size of the network in terms of nodes and edges, the number of time points in the traces, and the settings produced by the parameters that are varied, as discussed below. Malicious and botnet users are much more active than non-malicious ones; we designed it in this manner assuming that the former post new content only (they do not share) – since $prob_{m\_share} = 0$ and $prob_{b\_share} = 0$, respectively – and want to disseminate as much fake news articles as possible. With these

[7] We used the implementation available at: https://github.com/farkhor/PaRMAT.

Setting	Parameter Values			
	t_{max}	Variant of r_1	$prob_{memb}$	$prop_{mal}$
A	2	$r_{1.2}$	0.25	0.2
B	2	$r_{1.3}$	0.25	0.2
C	2	$r_{1.1}$	0.25	0.1
D	2	$r_{1.1}$	0.1	0.2
E	5	$r_{1.2}$	0.25	0.2
F	5	$r_{1.3}$	0.25	0.2

Setting	Shorthand Labels			
	Time	Detection level	#Bots	#Malicous
A	Short	Bold	MoreBots	MoreMal
B	Short	Cautious	MoreBots	MoreMal
C	Short	Boldest	MoreBots	FewMal
D	Short	Boldest	FewBots	MoreMal
E	Long	Bold	MoreBots	MoreMal
F	Long	Cautious	MoreBots	MoreMal

Fig. 4. Parameter values (left) and corresponding shorthand labels (right) for settings used in the evaluation; fixed parameters are set as described in Section 3.1.

settings, the probability of a malicious actor posting a fake news article at each time point is $prob_{m\_post} * prob_{m\_mal} = 0.3$ (the probability for botnet actors is also the same value $prob_{b\_post} * prob_{b\_mal} = 0.3$), while for a non-malicious one it is $prob_{nm\_post} * prob_{nm\_mal} = 0.005$ (though the latter might still unintentionally share a fake news article from their feeds).

Additionally, a set of other parameters were varied in order to widen the scope of the evaluation, creating six different settings, which we call A–F; see Fig. 4 (left) for details (the "Variant of r_1" column is explained in the next subsection). Likewise, Fig. 4 (right) provides shorthand labels for each setting according to the following set of distinctive properties:

- **Time**: This setting refers to the number of time steps of the diffusion process (temporal duration of the simulation, which is denoted with t_{max}). We consider two possible lengths of this simulation horizon: Short ($t_{max} = 2$) and Long ($t_{max} = 5$).
- **Detection level**: This property refers to the threshold for the global network condition in rule r_1, which corresponds to how sensitive the rule will be to the *trending* label. We consider three possible detection levels: Boldest (threshold 0.3, rule $r_{1.1}$), Bold (threshold 0.5, rule $r_{1.2}$), and Cautious (threshold 0.7, rule $r_{1.3}$).
- **#Bots**: Refers to the number of members in a botnet. In this case, we consider two possible values: FewBots ($prob_{memb} = 0.1$) and MoreBots ($prob_{memb} = 0.25$).
- **#Malicious**: Analogous to the previous property, this setting refers to the number of malicious users in the network. Here, we also consider two possible values: FewMal ($prop_{mal} = 0.1$) and MoreMal ($prop_{mal} = 0.2$).

To arrive at a descriptor for a setting, we simply concatenate each label; for example, the shorthand for setting A is "Short--Bold--MoreBots--MoreMal". This naming scheme is designed to help identify the effect of parameter variations and thus make comparisons between the results obtained for each setting (presented in Section 4).

In Section 3.3, we detail how BadBot is effectively used to generate the traces to be fed to NetDER within the experimental setting.

3.2. Instantiation of NetDER

We now provide the details of how the general NetDER architecture is implemented for our evaluation. The data ingestion module is tasked with creating the data at each time for the ontological database and the network database, based on the current state of the trace generated by the testbed. We have the following schema for the ontological database:

- *news*(N): N is a news article (from dataset [7], as described above);
- *category*(N, C): N corresponds to category C;
- *fn_level*(N, L): N is judged to be a fake news article with confidence level L—this is a wrapper for an external machine learning-based classifier (in our evaluation, we used the tool available at [10]);
- *early_poster*(U, N): user U was one of the first to post article N;
- *close*(U_1, U_2): users U_1 and U_2 posted the same article at the same time.

The structure of the network is obtained directly from the BadBot traces; we have node local labels *pref_category*($categ_1$)–*pref_category*($categ_5$) to represent the confidence that each category is the preferred one for each node; these labels are updated at each time point according to users' posting activities.

Fig. 5 gives a detailed overview of how the main parts of the NetDER architecture are instantiated for our empirical evaluation. We now provide details and intuitive descriptions of each part; the rules were written manually by domain experts, which in our case is a simple task given that we have access to the full details of how the testbed works. So, for the diffusion rules, having a good model means that we will have accurate simulations of how fake news spreads. However, note that the resulting models are not perfect given the complexity and stochastic nature of the environment.

Ontology Module Rules (top vignette in Fig. 5): Set of logical rules that drive the generation of hypotheses that will result in query answers (see below).[8]

The underlined predicate *fn_level* is a wrapper for a call to an external machine learning classifier that, given a news article, returns a level of confidence with which the article is believed to be fake news. Though in this evaluation we use the tool provided in [10], it can be replaced by any classifier (or set of classifiers) if desired; below we describe a variant of these rules that assumes a *perfect* classifier in order to see the effect that this component has on overall performance.

- r_1 has three variants according to different thresholds. Intuitively, the rule states that given a news article that the classifier detects as fake with confidence level L, create a hypothesis that the article is fake news. The rule only applies when the article's category is trending in the network (the intuition being that trending categories are more likely to yield fake news). The three variants correspond to different thresholds for the confidence and trending levels. r_2 is a simplification of r_1 that does not consider trending categories.
- r_3 generates hypotheses regarding who is responsible for disseminating fake news based on who is identified as an early poster. r_6 is a simpler variant that applies for any poster, not just early ones.
- r_4 states that if a user is responsible for at least two different fake news articles, then (s)he is malicious. r_7 is a simpler variant that only requires responsibility for one article.
- r_5 generates hypotheses for the existence of a botnet and its membership whenever two users post the same content at the same time. r_8 is a variant that replaces posting at the same time with a connection between the users (designed to be a low-quality version of r_5, since this is not how botnets work in the testbed).

8 Rules r_1–r_8 are *tuple-generating dependencies* (TGDs), while r_9 is an *equality-generating dependency* (EGD). We refer the interested reader to [1] for more details.

Ontology Module Rules: (*fn_level* is a wrapper for an external ML classifier)

$r_{1.1}$: $news(N) \wedge category(N, C) \wedge \underline{fn\_level}(N, L) \wedge (L > 0.2) \rightarrow hyp\_fakenews(N) : \langle trending(C), [0.3, 1]\rangle$

$r_{1.2}$: $news(N) \wedge category(N, C) \wedge \underline{fn\_level}(N, L) \wedge (L > 0.2) \rightarrow hyp\_fakenews(N) : \langle trending(C), [0.5, 1]\rangle$

$r_{1.3}$: $news(N) \wedge category(N, C) \wedge \underline{fn\_level}(N, L) \wedge (L > 0.2) \rightarrow hyp\_fakenews(N) : \langle trending(C), [0.7, 1]\rangle$

r_2 : $news(N) \wedge \underline{fn\_level}(N, L) \wedge L > 0.5 \rightarrow hyp\_fakenews(N)$

r_3 : $hyp\_fakenews(N) \wedge early\_poster(UID, N) \rightarrow hyp\_is\_resp(UID, N)$

r_4 : $hyp\_is\_resp(UID, N_1) \wedge hyp\_is\_resp(UID, N_2) \wedge (N_1 \neq N_2) \rightarrow hyp\_malicious(UID)$

r_5 : $hyp\_malicious(UID_1) \wedge hyp\_malicious(UID_2) \wedge closer(UID_1, UID_2) \rightarrow \exists B\ hyp\_botnet(B) \wedge member(UID_1, B) \wedge member(UID_2, B)$

r_6 : $hyp\_fakenews(N) \wedge posted(UID, N) \rightarrow hyp\_is\_resp(UID, N)$

r_7 : $hyp\_is\_resp(UID, N) \rightarrow hyp\_malicious(UID)$

r_8 : $hyp\_malicious(UID_1) \wedge hyp\_malicious(UID_2) \wedge (edge(UID_1, UID_2)) \rightarrow \exists B\ hyp\_botnet(B) \wedge member(UID_1, B) \wedge member(UID_2, B)$

r_9 : $member(UID, B_1) \wedge member(UID, B_2) \rightarrow B_1 = B_2$

Diffusion Module Rules:

$diff\_rule_1$: $pref\_category(C)_\top \overset{1}{\leftarrow} \top, \langle pref\_category(C), [1, 1]\rangle, if_1$

$diff\_rule_2$: $trending(C) \leftarrow (pref\_category(C)_\top, avg)$

$if_1(neigh, qua\_neigh, NI) =$

$$\begin{cases} [1, 1] \text{ if } |qua\_neigh|/|\{v \text{ s.t. } v \in neigh \wedge (v, \langle L, [1, 1]\rangle) \in NI\}| > 0.5 \\ [0, 1] \text{ otherwise} \end{cases}$$

The aggregation function *avg* is defined as $avg(B) = [l, u]$ where:

$$l = \sum_{[l_i,u_i]\in B}^{|B|} \frac{l_i}{|B|} \quad \text{and} \quad u = \sum_{[l_i,u_i]\in B}^{|B|} \frac{u_i}{|B|}$$

KB Variants: (cf. Figure 4 for details regarding variants of r_1 according to setting)

$KB_\alpha = \{r_{1.x}, r_2, r_3, r_4, r_5, r_9, diff\_rule_1, diff\_rule_2\}$

$KB_\beta = \{r_{1.x}, r_2, r_6, r_7, r_8, r_9, diff\_rule_1, diff\_rule_2\}$

$KB_{\alpha^*} = \{r_2^*, r_3, r_4, r_5, r_9, diff\_rule_1, diff\_rule_2\}$

(r_2^* is a modification of r_2 where the *fn_level* predicate is a perfect classifier.)

Queries:

$Q_{resp}(Y, Z) = hyp\_is\_resp(Y, Z) : [t_{max}, t_{max}]$

$Q_{mal}(M) = hyp\_malicious(M) : [t_{max}, t_{max}]$

$Q_{memb}(UID) = \exists B\ member(UID, B) \wedge hyp\_botnet(B) : [t_{max}, t_{max}]$

$Q_{fnA}(X) = hyp\_fakenews(X) : [t_{max}, t_{max}]$

$Q_{fnB}(N) = fn\_level(N, L) \wedge (L > 0.2) : [t_{max}, t_{max}]$

$Q_{fnC}(N) = fn\_level(N, L) \wedge (L > 0.5) : [t_{max}, t_{max}]$

Fig. 5. Main details of the instantiation of the NetDER architecture for the evaluation. See Section 3.2 for an intuitive description of each component.

- Finally, r_9 simply states that there exists a single botnet.

Diffusion Module Rules (second vignette in Fig. 5): Rule $diff\_rule_1$ is local, which actually represents a set of five rules (one for each news category in our evaluation). Such local rules estimate which category is the preferred one for each node by updating the $pref\_category(C)$ labels—the intervals in this case can take only two values: [0, 1] means uncertain likelihood of being preferred, and [1, 1] complete certainty. This is done via the if_1 influence function,[9] which takes the set of neighbors who posted an article of category C and, if that set represents more than half of the

[9] Note that "if_1" appears in the body of the rule without parameters—this is simply a reference to the function that should be used.

neighbors who posted something, then the label for category C is updated to [1, 1], meaning that category C is likely to be preferred by the node in the following time point.

The global rule $diff\_rule_2$ – also a set of five rules – simply takes an average of the $pref\_category(C)$ intervals (given the simplification described above, effectively their lower bounds) and assigns these values to the "trending" label obtained from $pref\_category(C)$.

KB Variants (third vignette in Fig. 5): We prepared three different knowledge bases with the objective of showing how the quality of the knowledge engineering effort affects the overall performance of the system:

- KB_α is comprised of the first five rules, which were designed to more closely mirror the way the world works (i.e., the way the data is generated by the BadBot testbed).
- KB_β replaces r_3–r_5 with r_6–r_8, which were designed to be weaker versions.
- $KB_{\alpha*}$ is a copy of KB_α except that the *fn_level* predicate is a perfect classifier (i.e., it does not suffer from false positives nor false negatives). The goal of having this variant is to gauge the impact of the machine learning component of the system, which is an external module in this evaluation.

Queries (bottom vignette in Fig. 5): we have six different queries. The first three correspond, respectively, to the three main detection tasks we wish to carry out: Responsible, Malicious, and Member. The second group of queries are geared toward detecting which articles correspond to fake news; this is a secondary baseline task that we briefly discuss in Section 4.

Note that such queries rely on the hypotheses generated by the ontology module rules (*hyp_is_resp*, *hyp_malicious*, and *hyp_botnet*). In the case of botnets, since we are interested in the members, we use an existential quantifier for the botnet itself and rely on the *member* predicate to obtain the results.

3.3. Ground truth and evaluation of performance

One of the main motivations behind the design and use of a testbed like BadBot is the need to have access to *ground truth*, since this is the basis for calculating metrics that allow to compare how well different tools perform. In particular, we need to be able to determine without uncertainty whether an answer to a query is a true positive (TP, the output item is part of the actual answer[10]), false positive (FP, the output item is not part of the actual answer), true negative (TN, the item is not output and is not part of the actual answer), or false negative (FN, the item is not output but is part of the actual answer). We now clarify the most important details regarding ground truth:

- A news article corresponds to *fake news* if the dataset classifies it as such.
- A user is *malicious* if BadBot marked it as such when generating the trace.
- A user u is *responsible* for a news article n if n is fake news, u posted n, and u is malicious,
- A user u is *member* of a botnet b if u was marked as such when generating the trace.

Note that ground truth is defined in terms of a given BadBot trace. The performance of a given NetDER KB on a query answering task is therefore defined in terms of the basic outcomes described above. Since it is possible that the same answer may appear in two (or more) time points, we count each answer *only once* (as TP, FP, TN, or FN). However, when an answer is counted as FN, it may become TP in a later time point, but again this is done only once for each answer. In summary, we consider answers that are *not given* to be false negatives, but the correct answer may come up when new information arrives and thus become TP.

The following algorithm describes the general workflow that we used in our experiments:

1. Set $new\_graph = true$
2. Repeat #*runs* times:
   (a) Generate a trace using BadBot with given parameters;
   (b) For each time point from 0 to t_{sim}:
   i. Issue query using the *one shot* NetDER strategy (execute chase → execute diffusion process → answer query);
   ii. Compute FPs, FNs, TPs, and TNs with respect to the ground truth and answers obtained in the previous step;
   (c) Calculate results for the trace;
   (d) Set $new\_graph = false$
3. Calculate results for the experiment.

This general algorithm is used to obtain all results reported in the next section, using the same traces to evaluate the three KB variants and focusing on three performance metrics:

- *Precision* and *Recall*: Defined as usual as $TP/(TP + FP)$ (fraction of obtained answers that are relevant to the query) and $TP/(TP + FN)$ (fraction of relevant answers that are in fact obtained), respectively.
  Though typically the F1 measure is also reported along with precision and recall (computed as the harmonic mean of the two), in our setting there are many instances in which it is undefined due to divisions by zero; there are ways to work around this issue, but none of them resulted in a satisfactory outcome since in general the results were artificially high. Given that the actual numeric results are not the focus of our evaluation – instead, we wish to show the effects of making changes to the environment and models – we chose not to report this metric.
- *Time to detect*: Number of time points in the trace between the first occurrence of an event and its incorporation into a query's answer. For instance, the posting of a fake news article by a malicious node and the addition of the corresponding *hyp_is_resp* atom. This metric captures a lower bound to *reaction time* by applications of the tool in smart policing efforts.

In the next section, we focus on analyzing the results of our experiments.

4. Results: Performance of three models in six settings

We first discuss the results for the three main tasks we tackle, and then finish by briefly mentioning the results for the auxiliary fake news detection task.

As context for these results, we point out that the main objectives in carrying out the experimental evaluation were: (i) to show that a variety of different settings can be captured in the testbed, and (ii) that different engineering decisions during model construction and the quality of ML classifiers that may be available have an impact on performance metrics for the three studied problems. In particular, settings A–F should not be interpreted as mapped to more or less realistic scenarios, but rather as variations showing how different aspects of the domain and problems impact performance. We will come back to these issues in the closing discussions in Section 6.

[10] By "actual answer" we refer to the set of answers provided by a perfect query answering mechanism having access to ground truth.

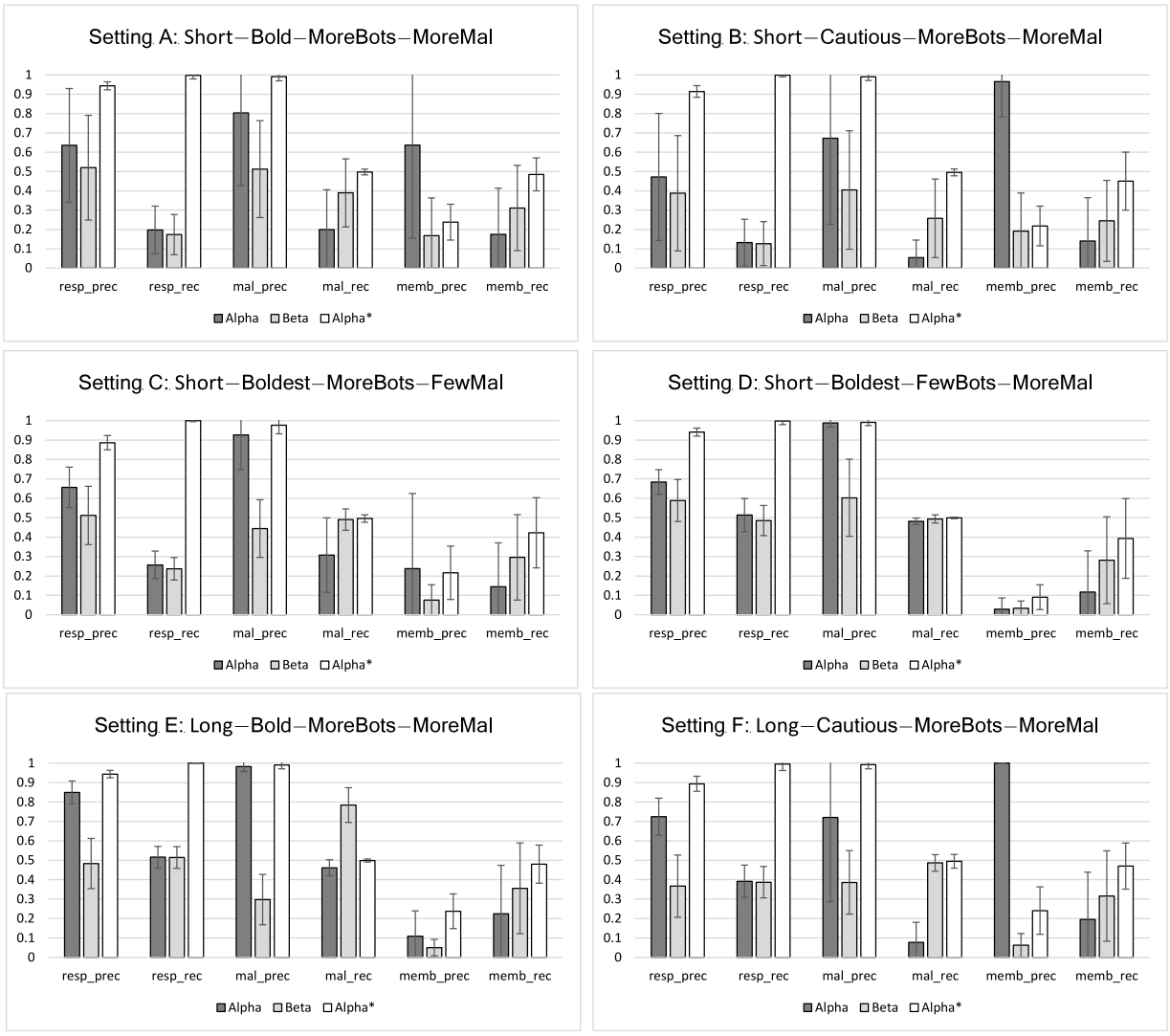

Fig. 6. Plots for average precision and recall (higher is better, error bars represent standard deviation) yielded for KB_α, KB_β, and KB_{α^*} for the three queries in Settings A–F. (Results for 100 runs; cf. Fig. 8 for statistical significance results).

4.1. Main tasks

Fig. 6 shows the results for precision and recall, and Fig. 7 those for time to detect; all plots correspond to averages and standard deviations obtained over 100 runs, where everything in the testbed is reset at the start of each run except for the graph structure (this is done in an effort to keep experimental variation manageable). The size of the error bars associated with many of the results obtained indicate that there is a non-trivial amount of variation in the environment, which is to be expected in complex problems like the ones being tackled. However, the number of runs performed allowed us to obtain highly statistically significant results in the vast majority of the cases, as can be seen in Fig. 8.

In the rest of this section, we will analyze these results from different perspectives. We begin with broad comparisons among the KB variants across all settings.

KB_{α^*} vs. Rest. As it might be expected, having access to a perfect classifier (one that does not suffer from false positives nor false negatives) yields notable performance improvements in comparison with the "regular" variants. However, some less-expected results were also obtained:

- RESPONSIBLE: Near-perfect precision, recall, and detection times across all settings.
- MALICIOUS: Near-perfect precision across all settings, but recall around 0.5, indicating the presence of false negatives. Perhaps rule r_4 is too restrictive, and some malicious nodes are missed because of the requirement to have two different *hyp_is_resp* atoms. Using only one (as done by r_7) might be a better option considering that the fake news predictor does not make any mistakes.
  Regarding detection times, we observe worse performance for Settings E and F, which correspond to those with longer simulation horizons. This is because the longer simulation influences fake news detection in KB_α and KB_β but has no influence in KB_{α^*} since the perfect classifier works in isolation. Then, KB_α and KB_β generate greater amounts of answers for Settings E and F, which leads to better chances for faster detections.
- MEMBER: Better recall and detection times than the rest. However, only better precision in two settings, which at first seems counterintuitive. However, considering the interaction of factors, our analysis is that – depending on the setting – the threshold of the global label in rule r_1 is too high, or t_{max} is too low. Since all members of the botnet post the same content at the same time, the categories of their

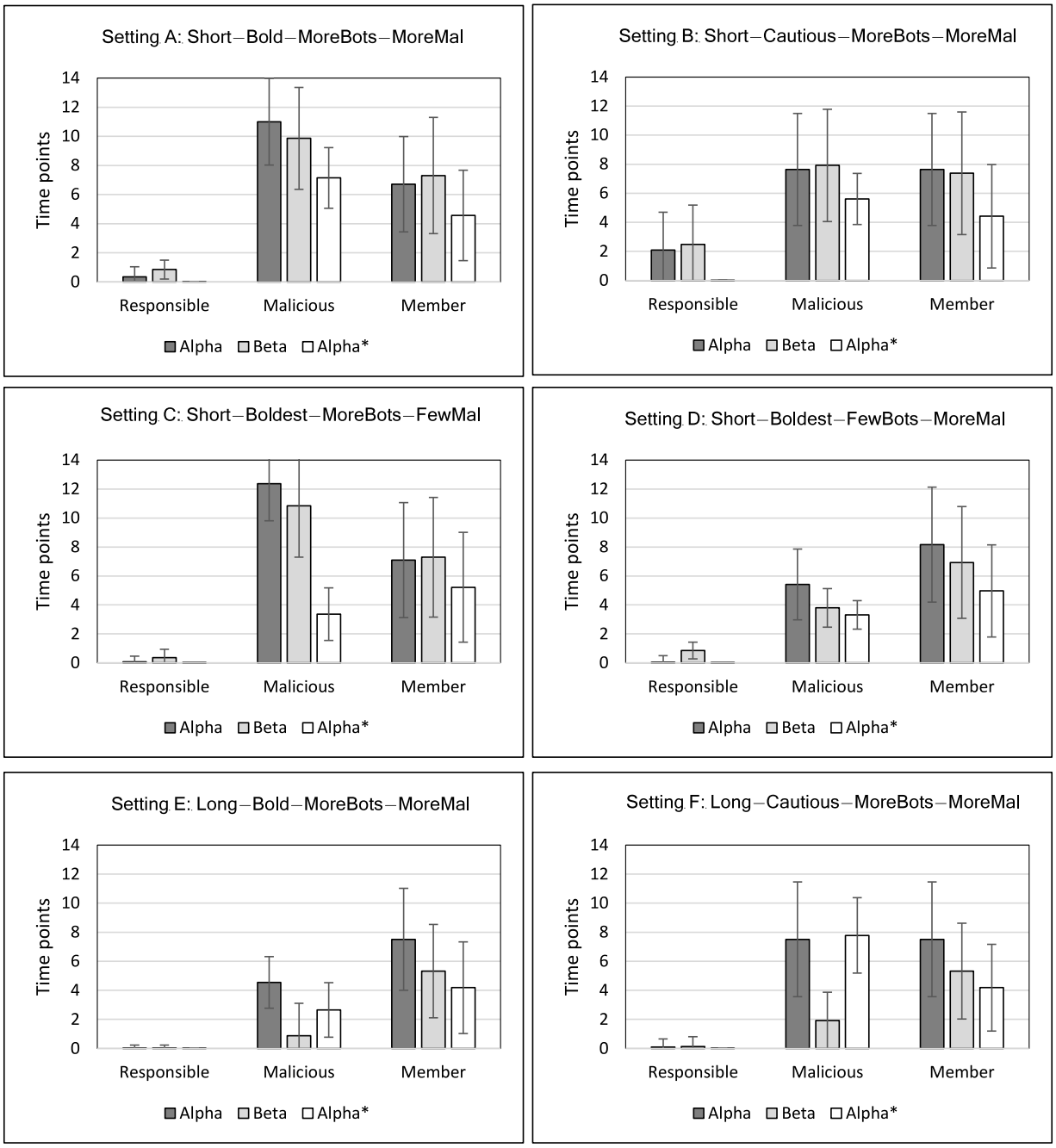

Fig. 7. Plots for average detection times (lower is better, error bars represent standard deviation) yielded for KB_{α}, KB_{β}, and KB_{α^*} for the three queries in Settings A–F (Results for 100 runs; cf. Fig. 8 for statistical significance results).

posts are the most likely to push the bound of the global label over the threshold, even when the simulation (diffusion process) is short. On the other hand, the simulation has no influence in KB_{α^*}.

KB_{α} vs. KB_{β}. The α variant was designed to better mimic the way traces are generated by BADBOT, while β uses weaker versions of several rules. This leads to worse precision but better recall, which corresponds to more false positives but fewer false negatives, as well as better detection times, across all evaluated settings.

We now move on to more specific comparisons, focusing on the parameter variations that gave rise to the different settings. We focus primarily on the performance of KB_{α} in order to provide more specific conclusions.

Setting A *vs. Setting* E. These two settings differ only in length of simulation horizon (t_{max}):

- RESPONSIBLE and MALICIOUS: Longer simulations yield better precision and recall, as well as shorter detection times.
- MEMBER: For this task, the reverse is true, a *shorter* simulation performs better. This is likely related to our analysis for KB_{α^*} vs. the rest, in which a shorter simulation can result in the categories of botnet members' posts becoming the most frequent.

Setting B *vs. Setting* F. These two settings also differ only in length of the simulations (t_{max}); compared with the previous pair, these two settings use a variant of rule r_1 with a higher threshold, which means that it is applied less frequently.

Setting	Query		(KB_α, KB_β)	$(KB_\alpha, KB_{\alpha^*})$	$(KB_\beta, KB_{\alpha^*})$
A	Responsible	Precision	**	**	**
		Recall	*ns*	**	**
	Malicious	Precision	**	**	**
		Recall	**	**	**
	Member	Precision	**	**	**
		Recall	**	**	**
B	Responsible	Precision	$p = 0.063$	**	**
		Recall	*ns*	**	**
	Malicious	Precision	**	**	**
		Recall	**	**	**
	Member	Precision	**	**	*ns*
		Recall	**	**	**
C	Responsible	Precision	**	**	**
		Recall	*	**	**
	Malicious	Precision	**	*	**
		Recall	**	**	*ns*
	Member	Precision	**	*ns*	**
		Recall	**	**	**
D	Responsible	Precision	**	**	**
		Recall	**	**	**
	Malicious	Precision	*ns*	**	**
		Recall	**	*	**
	Member	Precision	**	**	**
		Recall	**	**	**
E	Responsible	Precision	**	**	**
		Recall	*ns*	**	**
	Malicious	Precision	**	*	**
		Recall	**	**	**
	Member	Precision	**	**	**
		Recall	**	**	**
F	Responsible	Precision	**	**	**
		Recall	*ns*	**	**
	Malicious	Precision	**	**	**
		Recall	**	**	*ns*
	Member	Precision	**	**	**
		Recall	**	**	**

Setting	Query	(KB_α, KB_β)	$(KB_\alpha, KB_{\alpha^*})$	$(KB_\beta, KB_{\alpha^*})$
A	Resp.	**	**	**
	Mal.	**	**	**
	Memb.	*	**	**
B	Resp.	**	**	**
	Mal.	*ns*	**	**
	Memb.	*ns*	**	**
C	Resp.	**	**	**
	Mal.	**	**	**
	Memb.	*ns*	**	**
D	Resp.	**	**	**
	Mal.	**	**	**
	Memb.	*	**	**
E	Resp.	$p = 0.08$	**	**
	Mal.	**	**	**
	Memb.	**	**	$p = 0.092$
F	Resp.	**	**	**
	Mal.	**	*ns*	**
	Memb.	**	**	**

Fig. 8. Results for two-tailed two-sample unequal variance Student's t-tests for all pairs of comparable performance results plotted in Fig. 6 (*top*) and Fig. 7 (*bottom*). For each result, we use "**" to denote *highly significant* ($p < 0.01$), "*" to denote *significant* ($p < 0.05$), an explicit p-value for $0.05 \leq p < 0.1$, and "*ns*" to denote *not significant* ($p \geq 0.1$).

- Responsible and Malicious: Same relationship as in the previous pair, but much less marked for Malicious.
- Member: Precision is similar (notably, both yielded near-perfect results), and recall is slightly better for the longer simulation setting.

Detection times exhibited no significant differences between these two settings.

Setting A *vs. Setting* B. The only difference here is the variant of rule r_1, as mentioned above (B has a higher threshold):

- Responsible and Malicious: Higher precision and recall for the lower-threshold variant. Regarding time to detect, the same is true for Responsible, but for Malicious the higher-threshold variant performs better.
- Member: Setting B clearly outperforms A in this task; however, regarding detection times, setting A performs slightly better.

Setting C *vs. Setting* D. These two settings differ in that C has more botnet members, but less malicious users than D:

- Responsible and Malicious: Setting D generally outperforms C in both tasks, especially with respect to recall. Regarding detection times, the difference only lies in Malicious, for which D performs far better.
- Member: The reverse is true for this task, in which C performs better (mostly in terms of precision). Detection times are also slightly better in setting C.

Other notable observations. Overall, we can see that Member is the most difficult task, yielding precision and recall below 0.5 in all settings even when given access to a perfect fake news detector. We can mention two reasons for this. First, the rule that is specific to this task is not the best since it focuses on *pairs* of users; other alternatives should be explored, perhaps considering larger groups. Second, the information obtained from the simulation (diffusion process) is useful for this particular query, and KB_{α^*} does not use it; incorporating rules that analyze the most-spread topics might lead to better performance. Another general observation we can make is that when an excellent predictor is available, it may be more useful to adopt more relaxed rules.

As a final remark, we should say that depending on the application domain, the importance of false positives versus shorter detection times may vary greatly, and this should be taken into account. In general, we observed that KB_{α^*} has fewer false positives than KB_β, but the latter has shorter detection times. For example, if we are trying to identify people with criminal records at the border, a low false positive rate is required; on the other hand, if the task is to predict a cyber attack then there is typically limited time before it occurs and false positives may be addressed by manual inspection of suspicious cases, though alert fatigue may become an issue.

4.2. Fake news detection task

Here, we seek to identify which posted articles correspond to fake news. Even though this is primarily done via the *fn_level*

Setting	*fnA*				*fnB*				*fnC*			
	Precision		*Recall*		*Precision*		*Recall*		*Precision*		*Recall*	
	avg	*stdev*	*avg*	*stdev*	*avg*	*stdev*	*avg*	*stdev*	*avg*	*stdev*	*avg*	*stdev*
A	0.36	0.17	0.21	0.10	0.27	0.09	0.33	0.10	0.36	0.22	0.12	0.08
B	0.35	0.19	0.17	0.10	0.23	0.08	0.32	0.11	0.34	0.21	0.13	0.09
C	0.36	0.11	0.22	0.08	0.25	0.07	0.31	0.09	0.42	0.17	0.16	0.07
D	0.42	0.11	0.30	0.08	0.30	0.08	0.39	0.10	0.48	0.13	0.23	0.07
E	0.41	0.12	0.35	0.08	0.31	0.09	0.40	0.10	0.61	0.16	0.30	0.08
F	0.35	0.14	0.26	0.10	0.23	0.08	0.33	0.12	0.47	0.19	0.22	0.09

Fig. A.9. Results for the fake news detection task.

wrapper predicate (which, as mentioned, simply invokes an external machine learning-based classifier), we implemented four different NetDER rules (the three variants of r_1, plus r_2), as well as three different queries (Q_{fnA}, Q_{fnB}, and Q_{fnC}) designed to show how the variation of different thresholds and aspects of the logical derivation impact the results. Since this is a secondary task, we include the full results in the appendix (cf. Fig. A.9). In brief, query Q_{fnA} – which is based on a low-threshold NetDER rule plus another rule that leverages access to a network diffusion process – obtains a result between the two more extreme and simpler alternatives (Q_{fnB} and Q_{fnC} simply access the classifier's result and apply a lower and higher threshold, respectively).

Recall that the goal of this effort is not to obtain the highest possible performance metrics for fake news detection (or any other task), but rather show how different aspects of the system affect performance. Above we saw how three different variants of the NetDER system – including one with access to a perfect "oracle" classifier – performed in the more complex detection tasks.

5. Related work

In this section, we will first briefly touch upon the literature related to data and knowledge management, centering on the basic issues that arise when implementing a powerful model like NetDER instead of going into detail, which is outside the scope of this work. Then, we dedicate a larger discussion to related work on addressing malicious behavior in social platforms.

5.1. Data and knowledge management

The classical problems of data integration and data exchange in databases [11,12] and belief merging in the more general setting of Knowledge Representation and Reasoning [13,14] are at the core of the tasks that must be carried out by the data ingestion module, which must decide how to incorporate new data that arrives and may be in conflict with data already in place. The issue of trust in data sources must also be addressed; a powerful tool that can be applied here is argumentative reasoning, as done in [15], which evaluates reasons for and against different claims so that the best use is made of available information. Other, perhaps more basic, problems occur in settings where data integration is carried out without considering adequate integrity constraints (or, even when not explicitly integrated, are queried under a single virtual schema). Such problems are commonly known an *inconsistency* (where the data violates certain constraints) and *incoherence* (where the constraints themselves are impossible to satisfy). See [16] for a treatment of inconsistency in Datalog+/– knowledge bases (the underlying language used in NetDER rules), and [17,18] for recent works on incoherence.

Finally, in a more application-driven approach, [19] introduces a formalism designed to represent multiple social platforms in a single so-called *Network Knowledge Base*. Their goal is to model how information flows through networks; preliminary empirical evaluations with simple variants of this model have shown that it can be applied in the prediction of user reactions to Twitter feed content [20]. This approach could also be used in the NetDER data ingestion module as part of the implementation of the data model underlying the network diffusion module.

5.2. Malicious behavior in social platforms

This topic has been tackled by many researchers in different disciplines and – more recently – the large industry players behind the most popular social platforms. In particular, two companies that have taken action in this realm are Facebook (behind the social network of the same name, as well as Instagram and WhatsApp) [21] and Twitter [22]. It is becoming more increasingly apparent that industry actors – not only heavy hitters but also fringe players – need to explicitly address these issues [23], and that this deserves the same gravitas as other topics within the cybsersecurity realm [24,25].

On the academic research side, there are many lines of work that address a variety of basic problems, typically in an isolated manner. Recently, we have developed an approach to a problem we called *adversarial deduplication* [26], a variant of the classical databases problem of deduplicaction/entity resolution in which the typical "innocent mistake" assumption does not necessarily hold. Instead, actors use multiple identities to avoid being recognized by law enforcement, which leads to a more challenging scenario. Other related efforts are [27] and [28].

The research line leading up to the present paper continued in [29], where we proposed the use of *probabilistic* existential rules to generate hypotheses regarding duplicate actors, leveraging the value invention capability. This makes it possible to identify actors with specific kinds of behavior, and later refine the hypotheses when new information becomes available. The full NetDER architecture used here was presented in [1] in a general systems view manner, and then fully developed from a theoretical point of view in [6]. As mentioned in several parts of the present paper, the availability of datasets with ground truth is often a barrier to evaluating tools such as these in practice, which motivated the development of the BadBot testbed. One of the overarching goals of this line of work is to enhance logic-based formalisms with knowledge obtained from the use of machine learning tools, functioning in a human-in-the-loop fashion.

Sock puppets. An interesting problem related to the ones addressed here is that of detecting *sock puppets*, a kind of malicious behavior in social media in which identities are created under false pretenses; there have recently been many efforts dedicated to this problem (see, for instance, [30,31]). The problem can be seen as a special case of the ones addressed here, and the works mentioned – which for instance combine ML classifiers with community detection algorithms, leveraging features based on posting activity, text analysis, and social network structure – could be implemented within the NetDER model.

Fake news and bots. The issue of fake news is not a new phenomenon, but became popular in 2016 with the US and UK elections that led to the election of Donald Trump and Brexit, respectively. The basic detection problem refers to determining the veracity of news in the context of possible intentional deception [32,33]. Most research in this area focuses on linguistic analyses via natural language processing techniques, or network and behavior analysis. To the best of our knowledge, there are no other works that explicitly model the propagation of posts as diffusion processes.

Detecting *bots* in the context of social platforms is a basic task that touches upon most of the problems discussed above [34–36]. In particular, there are studies that suggest that fake news and social bots are closely related [2]. Examples of tools for automatic detection include [37,38]. An interesting work is that of [39], which focuses on Twitter bots and considers a large number of features, including information diffusion patterns. A problem related to bots is the so-called Sybil attack, in which systems based on peer-to-peer connections is attacked using false identities, seeking to bypass reputation-based trust mechanisms. Attacks are often performed both using bots and human-controlled accounts [40,41].

Cybersecurity in general. As mentioned above, there are many related problems in the (vastly) more general cybersecurity domain. Most efforts apply machine learning tools and techniques, for instance, to detect offering of products in hacker markets [42], predicting exploits [43] and enterprise cyber attacks [44], and identify at-risk software [45]. The latter is an example of the kind of development we propose, in which a combination of logic-based formalisms and information from classifiers, is effectively used to solve a given set of tasks.

Finally, there are many other application domains that would benefit from knowledge-based human-in-the-loop tools like the one developed and evaluated in this paper. Two examples that show this potential are automated health care [46] and detection of corruption in government agencies [47]. There is clearly much work to be done in this direction, but preliminary results are promising.

6. Conclusions and future work

In this work, we set out to implement and test a version of the recently proposed NetDER architecture for automatic generation of hypotheses; in particular, we were interested in developing and evaluating tools for detecting malicious behavior in social platforms. This has been generally recognized to be a formidable task not only because it is a multi-faceted problem, but also because it is nearly impossible to obtain open datasets with adequate ground truth in order to evaluate performance. We first developed a general testbed (making its source code publicly available) specifically designed to generate full traces of posting activity involving fake news, malicious actors, and botnets, and then carried out an extensive empirical evaluation of three variants of our model over six environmental settings in order to show the effects that the quality of knowledge engineering tasks, the quality of the underlying machine learning classifier used to detect fake news, and specific environmental conditions have on smart policing efforts in social platforms.

Even though the results we obtained show the feasibility of developing such tools, there are several limitations associated with our efforts (inherent to limitations of space and computational resources) that point to the need for further developments and evaluations. In particular, we would like to run experiments considering two main aspects: (i) a larger set of environmental settings—i.e., a wider variation of BadBot parameters, especially for graph size and the parameters that remained fixed in this paper; and (ii) more model variants—i.e., a wider variation of rules, especially for the diffusion process. Another aspect that needs to be further investigated is the impact of adding more tasks, and how models can be adapted to yield the best possible results depending on estimations of environmental features, such as user types, threat models, etc. Yet another necessary next step is to focus on reality-based scenarios, which involves choosing a specific social network, time period, set of users, and content of interest. This sort of study is of interest to move past the initial proof-of-concept approach taken in this paper, and additional challenges – like dealing with lack of directly-accessible complete data annotated with ground truth – will arise.

Finally, an important avenue of current and future work is developing different ways of deriving *explanations* for the query answers derived from our model. The logic-based rules defined in both the ontology reasoning and network diffusion modules already provide a solid starting point for doing this. We envision tagging results with summaries of how they were derived, and allowing human-in-the-loop interactions so that analysts are able to provide feedback to the system; this will also provide valuable information on the performance of the system, which can eventually learn from its mistakes (false positives and false negatives). As discussed briefly at the end of Section 4.1, striking an adequate balance between timely results and alert fatigue will depend primarily on the application domain.

CRediT authorship contribution statement

Jose N. Paredes: Software, Conceptualization, Methodology, Validation, Formal analysis, Investigation, Writing - original draft, Writing - review & editing. **Gerardo I. Simari:** Conceptualization, Methodology, Validation, Formal analysis, Investigation, Writing - original draft, Writing - review & editing, Supervision, Project administration, Funding acquisition. **Maria Vanina Martinez:** Conceptualization, Methodology, Validation, Formal analysis, Investigation, Writing - original draft, Writing - review & editing, Funding acquisition. **Marcelo A. Falappa:** Conceptualization, Resources, Writing - review & editing, Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was funded in part by Universidad Nacional del Sur (UNS), Argentina under grants PGI 24/N046 and PGI 24/ZN34, Secretaría de Investigación Científica y Tecnológica FCEN–UBA (RESCS-2020-345-E-UBA-REC), CONICET, Argentina under gran PIP 11220170100871CO, and Agencia Nacional de Promoción Científica y Tecnológica, Argentina under grants PICT-2018-0475 (PRH-2014-0007) and PICT-2016-0215.

Appendix. Further details

In this section we provide further details not included in the main text: precision and recall results for the *fnA*, *fnB*, and *fnC* tasks (Fig. A.9, and full numeric results for precision/recall and detection time for the three main tasks (Figs. A.10 and A.11, which correspond to the bar graphs in Figs. 6 and 7, respectively).

Setting	KB_α				KB_β				$KB_{\alpha*}$			
	Precision		*Recall*		*Precision*		*Recall*		*Precision*		*Recall*	
	avg	*stdev*	*avg*	*stdev*	*avg*	*stdev*	*avg*	*stdev*	*avg*	*stdev*	*avg*	*stdev*
A	0.64	0.29	0.20	0.12	0.52	0.27	0.17	0.10	0.94	0.02	1.00	0.02
B	0.47	0.33	0.13	0.12	0.39	0.30	0.13	0.11	0.91	0.03	1.00	0.01
C	0.66	0.10	0.26	0.07	0.51	0.15	0.24	0.06	0.89	0.04	1.00	0.00
D	0.68	0.06	0.51	0.09	0.59	0.11	0.49	0.08	0.94	0.02	1.00	0.02
E	0.85	0.06	0.52	0.06	0.48	0.13	0.51	0.06	0.94	0.02	1.00	0.00
F	0.72	0.10	0.39	0.08	0.37	0.16	0.39	0.08	0.89	0.04	1.00	0.03

Setting	KB_α				KB_β				KB_{α^*}			
	Precision		Recall		Precision		Recall		Precision		Recall	
	avg	*stdev*	*avg*	*stdev*	*avg*	*stdev*	*avg*	*stdev*	*avg*	*stdev*	*avg*	*stdev*
A	0.80	0.38	0.20	0.21	0.51	0.25	0.39	0.18	0.99	0.02	0.50	0.01
B	0.67	0.45	0.05	0.09	0.40	0.31	0.26	0.20	0.99	0.02	0.50	0.02
C	0.93	0.18	0.31	0.19	0.44	0.15	0.49	0.06	0.98	0.04	0.50	0.02
D	0.99	0.02	0.48	0.02	0.60	0.20	0.49	0.02	0.99	0.02	0.50	0.00
E	0.98	0.03	0.46	0.04	0.30	0.13	0.78	0.09	0.99	0.02	0.50	0.01
F	0.72	0.43	0.08	0.10	0.39	0.16	0.49	0.04	0.99	0.02	0.49	0.04

Setting	KB_α				KB_β				KB_{α^*}			
	Precision		Precision		Precision		*Recall*		*Precision*		Recall	
	avg	*stdev*	*avg*	*stdev*	*avg*	*stdev*	*avg*	*stdev*	*avg*	*stdev*	*avg*	*stdev*
A	0.64	0.48	0.18	0.24	0.17	0.20	0.31	0.22	0.24	0.09	0.49	0.09
B	0.97	0.18	0.14	0.22	0.19	0.20	0.24	0.21	0.22	0.10	0.45	0.15
C	0.24	0.39	0.14	0.23	0.08	0.08	0.30	0.22	0.22	0.14	0.42	0.18
D	0.03	0.06	0.12	0.21	0.03	0.04	0.28	0.22	0.09	0.06	0.39	0.21
E	0.11	0.13	0.23	0.25	0.05	0.04	0.35	0.23	0.24	0.09	0.48	0.10
F	1.00	0.00	0.20	0.24	0.06	0.06	0.32	0.23	0.24	0.12	0.47	0.12

Fig. A.10. Precision and recall results in numeric format for the Responsible (top), Malicious (middle), and Member (bottom) tasks (cf. Fig. 6). Averages and standard deviations calculated over 100 runs.

Setting	Task	KB_α		KB_β		KB_{α^*}	
		avg	*stdev*	*avg*	*stdev*	*avg*	*stdev*
A	Responsible	0.36	0.68	11	2.98	6.71	3.27
	Malicious	0.86	0.65	9.86	3.50	7.31	3.99
	Botnet Member	0.00	0.00	7.15	2.08	4.57	3.10
B	Responsible	2.09	2.60	7.64	3.86	7.64	3.86
	Malicious	2.48	2.71	7.92	3.87	7.38	4.22
	Botnet Member	0.00	0.00	5.61	1.76	4.42	3.55
C	Responsible	0.08	0.39	12.37	2.56	7.09	3.78
	Malicious	0.37	0.58	10.84	3.54	7.29	4.13
	Botnet Member	0.00	0.00	3.37	1.81	5.22	3.78
D	Responsible	0.05	0.45	5.42	2.44	8.16	3.97
	Malicious	0.86	0.58	3.8	1.33	6.93	3.85
	Botnet Member	0.00	0.00	3.31	0.98	4.97	3.18
E	Responsible	0.03	0.19	4.54	1.78	7.51	3.51
	Malicious	0.03	0.21	0.88	2.22	5.32	3.21
	Botnet Member	0.00	0.00	2.65	1.87	4.18	3.16
F	Responsible	0.09	0.56	7.51	3.95	7.51	3.95
	Malicious	0.13	0.67	1.92	1.95	5.32	3.30
	Botnet Member	0.00	0.00	7.78	2.59	4.18	2.98

Fig. A.11. Average *time to detect* results in numeric format for the Responsible, Malicious, and Member tasks (cf. Fig. 7) Averages and standard deviations calculated over 100 runs.

References

[1] J.N. Paredes, G.I. Simari, M.V. Martinez, M.A. Falappa, Netder: An architecture for reasoning about malicious behavior, Inform. Syst. Front. 23 (1) (2021) 185–201, http://dx.doi.org/10.1007/s10796-020-10003-w.

[2] C. Shao, G.L. Ciampaglia, O. Varol, A. Flammini, F. Menczer, The spread of fake news by social bots, 2017, pp. 96–104, ArXiv Preprint arXiv:1707.07592.

[3] R. Agarwal, S. Barve, S.K. Shukla, Detecting malicious accounts in permissionless blockchains using temporal graph properties, 2020, CoRR abs/2007.05169.

[4] M. Spagnuolo, F. Maggi, S. Zanero, Bitiodine: Extracting intelligence from the bitcoin network, in: N. Christin, R. Safavi-Naini (Eds.), Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers, in: Lecture Notes in Computer Science, Vol. 8437, Springer, 2014, pp. 457–468.

[5] A. Calì, G. Gottlob, T. Lukasiewicz, A general datalog-based framework for tractable query answering over ontologies, J. Web Semant. 14 (2012) 57–83.

[6] J.N. Paredes, G.I. Simari, M.V. Martinez, M.A. Falappa, Combining existential rules with network diffusion processes for automated generation of hypotheses, Under Rev. (2021).

[7] W.Y. Wang, "Liar, liar pants on fire": A new benchmark dataset for fake news detection, in: R. Barzilay, M. Kan (Eds.), Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Association for Computational Linguistics, 2017, pp. 422–426.

[8] D. Chakrabarti, Y. Zhan, C. Faloutsos, R-MAT: A recursive model for graph mining, in: Proceedings of the Fourth SIAM International Conference on Data Mining, Lake Buena Vista, Florida, USA, April 22-24, 2004, pp. 442–446.

[9] F. Khorasani, R. Gupta, L.N. Bhuyan, Scalable SIMD-efficient graph processing on GPUs, in: Proceedings of the 24th International Conference on Parallel Architectures and Compilation Techniques, PACT '15, 2015, pp. 39–50.

[10] C. Rogerio, Fake news detector API, 2018, https://github.com/fake-news-detector/fake-news-detector/tree/master/api/#json-api-endpoints.

[11] P.G. Kolaitis, Reflections on schema mappings, data exchange, and metadata management, in: Proc. ACM SIGMOD/PODS, ACM, 2018, pp. 107–109.

[12] R.J. Miller, Open data integration, Proc. VLDB Endowment 11 (12) (2018) 2130–2139.

[13] S. Konieczny, R. Pino Pérez, Logic based merging, J Physiol (London) 40 (2) (2011) 239–270.

[14] M.A. Falappa, G. Kern-Isberner, M.D.L. Reis, G.R. Simari, Prioritized and non-prioritized multiple change on belief bases, J. Philos. Logic 41 (1) (2012) 77–113.

[15] M.A. Falappa, A.J. García, G. Kern-Isberner, G.R. Simari, Stratified belief bases revision with argumentative inference, J Physiol (London) 42 (1) (2013) 161–193.

[16] T. Lukasiewicz, M.V. Martinez, G.I. Simari, Inconsistency handling in datalog+/– ontologies, in: Proc. ECAI, 2012, pp. 558–563.

[17] C.A.D. Deagustini, M.V. Martinez, M.A. Falappa, G.R. Simari, Datalog+/–ontology consolidation, J. Artificial Intelligence Res. 56 (2016) 613–656.

[18] C.A.D. Deagustini, M.V. Martinez, M.A. Falappa, G.R. Simari, How does incoherence affect inconsistency-tolerant semantics for datalog+/–? Ann. Math. Artif. Intell. 82 (1–3) (2018) 43–68.

[19] F.R. Gallo, G.I. Simari, M.V. Martinez, M.A. Falappa, N.A. Santos, Reasoning about sentiment and knowledge diffusion in social networks, IEEE Internet Comput. 21 (6) (2017) 8–17.
[20] F.R. Gallo, G.I. Simari, M.V. Martinez, M.A. Falappa, Predicting user reactions to Twitter feed content based on personality type and social cues, Future Gener. Comput. Syst. 110 (2019) 918–930.
[21] Facebook, May 2020 Coordinated Inauthentic Behavior Report, Facebook, Inc., 2020, https://about.fb.com/news/2020/06/may-cib-report/.
[22] Y. Roth, N. Pickles, Updating our approach to misleading information, Twitter Blog (2020) https://blog.twitter.com/en_us/topics/product/2020/updating-our-approach-to-misleading-information.html.
[23] C. François, Actors, behaviors, content: A disinformation ABC, Algorithms (2020).
[24] G.I. Simari, From data to knowledge engineering for cybersecurity, in: IJCAI, 2019, pp. 6403–6407.
[25] A. Bruns, G. Zhu, Like a Virus: The Coordinated Spread of Coronavirus Disinformation, Centre for Responsible Technology, the Australia Institute, 2020, https://apo.org.au/node/305864.
[26] J.N. Paredes, G.I. Simari, M.V. Martinez, M.A. Falappa, First steps towards data-driven adversarial deduplication, Information 9 (8) (2018) 189.
[27] P. Jain, P. Kumaraguru, A. Joshi, @ i seek'fb. me': Identifying users across multiple online social networks, in: Proceedings of the 22nd International Conference on World Wide Web, ACM, 2013, pp. 1259–1268.
[28] A. Malhotra, L. Totti, W. Meira Jr, P. Kumaraguru, V. Almeida, Studying user footprints in different online social networks, in: Proc. ASONAM, IEEE Computer Society, 2012, pp. 1065–1070.
[29] J.N. Paredes, M.V. Martinez, G.I. Simari, M.A. Falappa, Leveraging probabilistic existential rules for adversarial deduplication, in: Proceedings of PRUV@IJCAR 2018, CEUR-WS, 2018.
[30] S. Kumar, J. Cheng, J. Leskovec, V. Subrahmanian, An army of me: Sockpuppets in online discussion communities, in: Proceedings of WWW, International World Wide Web Conferences Steering Committee, 2017, pp. 857–866.
[31] Z. Yamak, J. Saunier, L. Vercouter, Sockscatch: Automatic detection and grouping of sockpuppets in social media, Knowl.-Based Syst. 149 (2018) 124–142.
[32] K. Shu, A. Sliva, S. Wang, J. Tang, H. Liu, Fake news detection on social media: A data mining perspective, ACM SIGKDD Explor. Newsl. 19 (1) (2017) 22–36.
[33] N.J. Conroy, V.L. Rubin, Y. Chen, Automatic deception detection: Methods for finding fake news, Proc. Assoc. Inform. Sci. Technol. 52 (1) (2015) 1–4.
[34] M. Benigni, K.M. Carley, From tweets to intelligence: Understanding the islamic jihad supporting community on Twitter, in: K.S. Xu, D. Reitter, D. Lee, N. Osgood (Eds.), Social, Cultural, and Behavioral Modeling, Springer International Publishing, Cham, 2016, pp. 346–355.
[35] N. Abokhodair, D. Yoo, D.W. McDonald, Dissecting a social botnet: Growth, content and influence in Twitter, in: Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing, ACM, 2015, pp. 839–851.
[36] E. Ferrara, O. Varol, C. Davis, F. Menczer, A. Flammini, The rise of social bots, Commun. ACM 59 (7) (2016) 96–104.
[37] M.C. Benigni, K. Joseph, K.M. Carley, Online extremism and the communities that sustain it: Detecting the ISIS supporting community on Twitter, PLoS One 12 (12) (2017).
[38] M.C. Benigni, K. Joseph, K.M. Carley, Bot-ivistm: Assessing information manipulation in social media using network analytics, in: N. Agarwal, N. Dokoohaki, S. Tokdemir (Eds.), Emerging Research Challenges and Opportunities in Computational Social Network Analysis and Mining, Springer International Publishing, Cham, 2019, pp. 19–42.
[39] C.A. Davis, O. Varol, E. Ferrara, A. Flammini, F. Menczer, BotOrNot: A system to evaluate social bots, in: Proceedings of the 25th International Conference Companion on World Wide Web, International World Wide Web Conferences Steering Committee, 2016, pp. 273–274.
[40] G. Noh, Y.-m. Kang, H. Oh, C.-k. Kim, Robust sybil attack defense with information level in online recommender systems, Expert Syst. Appl. 41 (4) (2014) 1781–1791.
[41] A. Kumar, D. Garg, P. Singh, Clustering approach to detect profile injection attacks in recommender system, Int. J. Comput. Appl. 166 (6) (2017) 7–11.
[42] E. Marin, A. Diab, P. Shakarian, Product offerings in malicious hacker markets, in: 2016 IEEE Conference on Intelligence and Security Informatics (ISI), IEEE, 2016, pp. 187–189.
[43] N. Tavabi, P. Goyal, M. Almukaynizi, P. Shakarian, K. Lerman, Darkembed: Exploit prediction with neural language models, in: Thirty-Second AAAI Conference on Artificial Intelligence, AAAI Press, 2018.
[44] S. Sarkar, M. Almukaynizi, J. Shakarian, P. Shakarian, Predicting enterprise cyber incidents using social network analysis on the darkweb hacker forums, 2018, CoRR abs/1811.06537.
[45] E. Nunes, P. Shakarian, G.I. Simari, At-risk system identification via analysis of discussions on the darkweb, in: 2018 APWG Symposium on Electronic Crime Research (ECrime), IEEE, 2018, pp. 1–12.
[46] A. Davoudi, K.R. Malhotra, B. Shickel, S. Siegel, S. Williams, M. Ruppert, E. Bihorac, T. Ozrazgat-Baslanti, P.J. Tighe, A. Bihorac, et al., Intelligent ICU for autonomous patient monitoring using pervasive sensing and deep learning, Sci. Rep. 9 (1) (2019) 1–13.
[47] R.B. Velasco, Identifying corruption risk in Brazil: New measures for effective oversight, in: R.I. Rotberg (Ed.), Corruption in Latin America: How Politicians and Corporations Steal from Citizens, Springer International Publishing, Cham, 2019, pp. 57–91, http://dx.doi.org/10.1007/978-3-319-94057-1_3.

Jose N. Paredes is a doctoral candidate at Universidad Nacional del Sur. His research interests are centered in knowledge representation and reasoning, with a special focus on reasoning about malicious behavior in cybersecurity domains. Contact him at: jose.paredes@cs.uns.edu.ar.

Gerardo I. Simari is an assistant professor in the Department of Computer Science and Engineering, Universidad Nacional del Sur, a researcher in the Institute for Computer Science and Engineering (Universidad Nacional del Sur–CONICET), and adjunct faculty at the School of Computing, Informatics, and Decision Systems Engineering (CIDSE), Arizona State University. His research interests include reasoning under uncertainty, preferences, social knowledge, and databases. He holds a Ph.D. from University of Maryland College Park. Contact him at: gis@cs.uns.edu.ar.

Maria Vanina Martinez is an assistant professor in the Department of Computer Science, Universidad de Buenos Aires, and a researcher at the Institute for Computer Science (Universidad de Buenos Aires–CONICET). Her research interests are at the intersection of knowledge representation and reasoning and database theory, with a special focus on reasoning with inconsistent and uncertain knowledge. She holds a Ph.D. from University of Maryland College Park. Contact her at: mvmartinez@dc.uba.ar.

Marcelo A. Falappa is an associate professor and dean of the Department of Computer Science and Engineering, Universidad Nacional del Sur, and a researcher at the Institute for Computer Science and Engineering (Universidad Nacional del Sur–CONICET). His research interests are centered in belief revision and argumentation-based reasoning. He holds a doctorate from Universidad Nacional del Sur. Contact him at: mfalappa@cs.uns.edu.ar.