Foundations of Languages for Interpretability and Bias Detection

Pablo Barceló[1,2], **Jorge Pérez**[2,3], **Bernardo Subercaseaux**[2,3]

[1] Institute for Mathematical and Computational Engineering, PUC-Chile
[2] Department of Computer Science, Universidad de Chile
[3] Millennium Institute for Foundational Research on Data, Chile
pbarcelo@ing.puc.cl, [jperez,bsuberca]@dcc.uchile.cl

Abstract

Despite the growing interest in interpretability for machine learning models, there seems to be a gap between the world of researchers and that of practitioners and data scientists. Declarative languages, tailored for interactive interpretability and bias detection of models, could be a step towards shortening such a gap. Based on requirements often posed over interpretability tasks, it is desirable for such languages to be interactive, model agnostic, faithful to the models, and have clear and well-understood semantics and complexity. A reasonable way to meet several of these requirements is by having a language strongly rooted in logic. As a first step towards the design of such language, we study how interpretability and bias detection queries can be expressed in first-order logic (FO) –arguably, one of the most prominent logics in different areas of computer science– over a suitable encoding of machine learning models. We then study the computational complexity of evaluating FO formulae over linear-based, tree-based and deep models. Evaluation is shown to be easier for decision trees and perceptrons than it is for deep neural networks, which seems validated by folklore assumptions of the field. Finally, we discuss possible extensions to the underlying logical structure and how they enhance expressiveness, as well as possible directions towards tractability.

1 Introduction

While research in methods for interpretability and bias detection has seen a significant increase in recent years [4], it seems that there is still a gap between the research community producing such methods and the community of data scientists and practitioners they are designed for [18, 20]. Although the exact extent of this gap remains to be determined [18], a line of research studying its nature and opportunities for growth seems to be emerging [18, 20]. An important concern in this respect is how to make the interpretation, auditing, and bias detection of machine learning processes as accessible as possible for non-experts. In particular, there is an increasing need for user-friendly interpretability and bias detection toolkits.

An interesting example of a user-friendly toolkit for bias and fairness detection is that of *Aequitas* [28], which automatically creates reports for users, based on labeled and predicted data, that include several metrics of bias and fairness. Each of such metrics can be understood as the answer to a particular *query* about the model to be interpreted. Furthermore, several well-defined queries about a model's behavior, or even particular decisions made by a model, can be found in the literature [11]. Popular methods to enhance interpretability, such as *SHAP* or Shapley values [23], can also be thought of as answers to a particular interpretability query.

Preprint. Under review.

The abundance of particular interpretability and bias detection queries in the literature suggests that one could aim for a general *query language*, tailored for interpretability and bias detection tasks, in which such queries can be declaratively expressed. This is of course reminiscent of the path many other areas in computer science have followed, in particular by using languages rooted in formal logic. For instance, in data management many formalisms based on first-order logic, or some of its extensions, have been developed for posing queries over different data models [1]; in knowledge representation the family of description logics is routinely applied for modeling application domains and extracting information over such models [5]; and in the area of model checking suitable temporal logics serve as bug-detection languages for specifying undesired behaviors of systems [9]. One of the advantages of this approach is that logics have a well-defined syntax and semantics, which typically facilitates the optimization and theoretical study of the computational complexity of the queries expressed in the language.

Several authors have advocated over the last years that logic should also have a prominent role in the context of interpretability for machine learning models [7, 10]. We strongly believe, in fact, that logic has to be key in the development of languages for interpretability and bias detection tasks. The reason is that logic-based languages help in establishing several good properties that seem to be desirable in this context. These properties include:

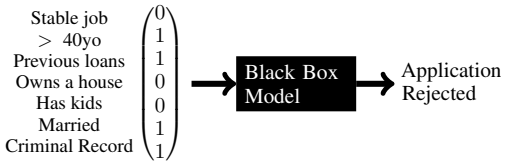
- **Faithfulness**: A language must only give answers that are faithful to the model, meaning that they can be mathematically verified, and are produced rigorously from the properties of the considered model, rather than by a learning process [27].
- **Interactivity**: Following [35], interpretation is not to be understood as a result but rather a process in which users interact with the system in order to interpret it and progressively gain understanding and trust.
- **Clear syntax and semantics**: Such a language must be well understood. It must be perfectly clear, in each interaction with the language, what exactly is that the user is querying, and what exactly is that the results retrieved mean.
- **Model agnosticism**: Considering the variety of models and architectures that are present in Machine Learning, it is desirable for such a system to allow interpretation (maybe to different degrees) of any model or architecture [26].

For logical languages, faithfulness can be obtained from the, provably correct, algorithms for query evaluation. Interactivity stems from the fact that the user is allowed to ask repeated queries, and design queries upon the answers obtained to the previous ones. A clear syntax and semantics is a *sine qua non* condition for a logical language to be accepted as such. Finally, model agnosticism comes from the fact that both at the language and logic level, the user refers to the model as an abstract entity that gives answers to inputs, without dealing with its internals. Therefore, any query can be applied to any underlying model of the same dimension.

Our paper aims to take initial steps in the direction of a logic-based language for interpretability and bias detection tasks, by identifying an underlying logic capable of expressing many natural interpretability and bias detection queries, and formally studying its complexity of evaluation. As an overview of our proposal consider the case of a bank using a binary model to judge applications for loans. Figure 1a illustrates the problem with concrete features, and Figure 1b presents an example of a concrete interactive syntax. In Figure 1b after loading and exploring the model saved in a file `"mlp.np"`, the first interaction asks whether the model could give a loan to a person who is married and does not have kids. We next show intuitively how can we formalize this interaction in a logical language.

Besides considering binary features as inputs, our formal setting considers also a third value, $\perp$, to represent that some feature is undefined (can take any value). We consider a binary logical predicate $\boldsymbol{x} \subseteq \boldsymbol{y}$ allowing to state when an instance $\boldsymbol{y}$ potentially fills some of the undefined features of another instance $\boldsymbol{x}$. For example $(1\,0\,\perp) \subseteq (1\,0\,1)$, but $(1\,0\,\perp) \not\subseteq (1\,1\,1)$. Moreover we consider a unary predicate, $\textsc{Positive}(\boldsymbol{x})$, that holds whenever an input instance $\boldsymbol{x}$ is classified with 1 by the model. These two are the only logical predicates considered in our initial analysis. With these, the first interaction in Figure 1b can be formalized as the following first-order logic (FO) sentence over the mentioned predicates

$$\exists \boldsymbol{x} \Big(\textsc{Positive}(\boldsymbol{x}) \wedge (\perp\ \perp\ \perp\ \perp\ 0\ 1\ \perp) \subseteq \boldsymbol{x} \Big) \tag{1}$$



(a) Diagram of a particular loan decision.

```
> load("mlp.np");
> show features;
(stableJob, >40yo, prevLoan, ownsHouse,
hasKids, isMarried, crimRecord)

> Exists? positive instance
    where isMarried = 1 and hasKids = 0
true

> Exists? positive instance
    where isMarried = 0 and hasKids = 1
false
```

(b) Example of a possible concrete syntax for a language tailored for interpretability queries.

Figure 1: Example of an unethical bank uses a model to decide whether to accept loan applications by using binary features like "does the requester has a stable job" and "are they older than 40".

As our main conceptual contribution we formalize the above intuition by encoding models as logical structures over which FO queries can be posed (Section 2). We also show that a big set of interpretability and bias detection queries already considered in the literature can be naturally expressed as FO sentences over the encoded models (Section 3). In terms of our technical contribution, we provide a preliminary study of the computational complexity of evaluating queries as FO formulas over the logical structures we use to encode models (Section 4). This analysis is parameterized by the length of the queries and considers three concrete classes of models: linear-based models, tree-based models, and multi-layer perceptrons (MLPs). Our complexity results reaffirm the common wisdom that evaluating queries largely depends on the underlying model, and extend previous formal results stating that interpretability and bias detection queries are easier to solve, from a complexity point of view, for models traditionally considered interpretable (as linear- and tree-based) than for models traditionally deemed opaque (as MLPs) [6]. As our results show there are many natural scenarios in which evaluating interpretability queries is intractable. Thus we also explore (in Section 5) two potential directions for coping with intractability in practical settings.

2 Models and Logical Structures

Models and instances. We consider an abstract definition of a *model* $\mathcal{M}$ simply as a Boolean function $\mathcal{M} : \{0,1\}^n \to \{0,1\}$. That is, we focus on binary classifiers with Boolean input features. A class of models is just a way of grouping models together. An *instance* is a vector in $\{0,1\}^n$ and represents a possible input for a model. The instances $(t_1, \ldots, t_n) \in \{0,1\}^n$ for which $\mathcal{M}(t_1, \ldots, t_n) = 1$ are called *positive*.

A *partial instance* is a vector in $\{0,1,\bot\}^n$, with $\bot$ intuitively representing "undefined" components. A partial instance $\boldsymbol{x} \in \{0,1,\bot\}^n$ represents, in a compact way, the set of all instances in $\{0,1\}^n$ that can be obtained by replacing undefined components in $\boldsymbol{x}$ with values in $\{0,1\}$. We call these the *completions* of $\boldsymbol{x}$.

Due to space constraints, we assume a general background with first-order logic (FO). Any reference textbook on the subject can be consulted otherwise; cf., [19].

Logical structures for encoding models and instances Consider a fixed model $\mathcal{M}$ and let n be its input size. We will define a logical structure $\mathfrak{A}$ on top of $\mathcal{M}$ following the ideas introduced in our example in the introduction of the paper. Let $U = \{0,1,\bot\}^n$ be the set of (partial) instances of dimension n, this will be the universe of $\mathfrak{A}$. The vocabulary of $\mathfrak{A}$ consists of constant symbols $\mathbf{c}_1, \ldots, \mathbf{c}_{|U|}$, for each element in U, the binary predicate symbol $\subseteq$ and the unary predicate symbol POSITIVE. We can now properly define $\mathfrak{A}$ as the following structure over σ:

$$\mathfrak{A} = \langle U, \{\mathbf{c}_1^{\mathfrak{A}}, \ldots, \mathbf{c}_{|U|}^{\mathfrak{A}}\}, \{\subseteq^{\mathfrak{A}}, \textsc{Positive}^{\mathfrak{A}}\}\rangle$$

The different components of $\mathfrak{A}$ are to be interpreted as follows. First, the constants $\mathbf{c_i}$ are simply interpreted as their corresponding element in U, that is, $\mathbf{c_i}^{\mathfrak{A}} = U_i$. Then, $\subseteq$ is a binary predicate whose interpretation includes all pairs $\boldsymbol{x}, \boldsymbol{y}$ such that for every component i it holds that either $x_i = \perp$ or $x_i = y_i$. The interpretation of the unary predicate POSITIVE includes all instances that are positive for $\mathcal{M}$.

Our proposal for a language is simply FO over the above described structures. That is we allow the use of quantifiers $\forall$ and $\exists$, conectors $\wedge$, $\vee$, $\rightarrow$ and constants and variables to construct formulas over the base predicates $\subseteq$ and POSITIVE. For instance, Equation 1 is a query in our language. The semantics of the language is directly inherited from that of FO.

Remark It is important to observe that the structure $\mathfrak{A}$ that represents model $\mathcal{M}$ contains all possible instances of $\mathcal{M}$ in its universe, and thus its size in exponential in the number of features of $\mathcal{M}$. This, of course, precludes the possibility of building $\mathfrak{A}$ in general. For the query evaluation problem, however, we do not assume $\mathfrak{A}$ to be given: the input consists only of $\mathcal{M}$ and the query to be evaluated over $\mathfrak{A}$. The study of the complexity of this problem then involves understanding when is that it is possible to evaluate the query in an "on-the-fly" fashion, i.e., *without* explicitly constructing $\mathfrak{A}$.

3 Selected interpretability and bias detection queries

We now show how the structure presented in Section 2 allows for encoding some particular queries that have already been linked to interpretability and bias detection [11, 30]. As an initial example, we consider the concept of a *Sufficient Reason* [11] for an instance $\boldsymbol{x}$ and a model $\mathcal{M}$, which intuitively consists of a subset of the features of $\boldsymbol{x}$ that explains its classification given by $\mathcal{M}$.

Definition 1 (Sufficient Reason)**.** *Given an instance $\boldsymbol{x}$ and a model $\mathcal{M}$, a* sufficient reason for $\boldsymbol{x}$ with respect to $\mathcal{M}$ *is a partial instance $\boldsymbol{y}$, such that $\boldsymbol{x}$ is a completion of $\boldsymbol{y}$ and every possible completion $\boldsymbol{x}'$ of $\boldsymbol{y}$ satisfies $\mathcal{M}(\boldsymbol{x}') = \mathcal{M}(\boldsymbol{x})$.*

Sufficient reasons may contain superfluous information. Consider for example that any instance $\boldsymbol{x}$ is by definition a sufficient reason for itself, yet we would not consider it a good explanation for its own verdict. In order to obtain more succinct sufficient reasons we consider the following definition, that strips sufficient reasons from superfluous information.

Definition 2 (Minimal Sufficient Reason)**.** *Given an instance $\boldsymbol{x}$ and a model $\mathcal{M}$, a* minimal sufficient reason for $\boldsymbol{x}$ with respect to $\mathcal{M}$ *is a partial instance $\boldsymbol{y}$, such that $\boldsymbol{y}$ is a sufficient reason for $\boldsymbol{x}$, but no proper subset $\boldsymbol{z}$ of $\boldsymbol{y}$ is a sufficient reason for $\boldsymbol{x}$.*

The notion of *Minimal Sufficient Reason* is analogous to that of a *Prime Implicant*, which has been amply studied in logic [17, 33].

We now show how these queries can be defined with the logic presented. We start by defining some auxiliary formulas that are useful for writing more sophisticated queries.

$$\textsc{Full}(\boldsymbol{x}) \equiv \forall \boldsymbol{z}(\boldsymbol{x} \subseteq \boldsymbol{z} \rightarrow \boldsymbol{x} = \boldsymbol{z}) \tag{2}$$

$$\textsc{Comp}(\boldsymbol{x}, \boldsymbol{y}) \equiv (\boldsymbol{y} \subseteq \boldsymbol{x}) \wedge \textsc{Full}(\boldsymbol{x}) \tag{3}$$

$$\textsc{SameClass}(\boldsymbol{x}, \boldsymbol{y}) \equiv \textsc{Full}(\boldsymbol{x}) \wedge \textsc{Full}(\boldsymbol{y}) \wedge \big(\textsc{Positive}(\boldsymbol{x}) \leftrightarrow \textsc{Positive}(\boldsymbol{y})\big) \tag{4}$$

The first formula distinguishes between partial instances in U and *full* instances (that can be passed as inputs to a model). The second one describes completions and third one describes instances that are classified the same by the underlying model.

We can now make a formula that checks whether a partial instance $\boldsymbol{y}$ is a sufficient reason for an instance $\boldsymbol{x}$, and then add the minimality requirement to obtain a minimal sufficient reason.

$$\textsc{SR}(\boldsymbol{x}, \boldsymbol{y}) \equiv \textsc{Comp}(\boldsymbol{x}, \boldsymbol{y}) \wedge \forall \boldsymbol{z}\big(\textsc{Comp}(\boldsymbol{z}, \boldsymbol{y}) \rightarrow \textsc{SameClass}(\boldsymbol{x}, \boldsymbol{z})\big) \tag{5}$$

$$\textsc{mSR}(\boldsymbol{x}, \boldsymbol{y}) \equiv \textsc{SR}(\boldsymbol{x}, \boldsymbol{y}) \wedge \forall \boldsymbol{z}\big(\textsc{SR}(\boldsymbol{x}, \boldsymbol{z}) \wedge \boldsymbol{y} \subseteq \boldsymbol{z} \rightarrow \boldsymbol{y} = \boldsymbol{z}\big) \tag{6}$$

3.1 Bias detection

The concepts of bias and unfairness are multidimensional, and require multidisciplinary perspectives and a constant dialogue with the social sciences [25]. In order to provide a self-contained example,

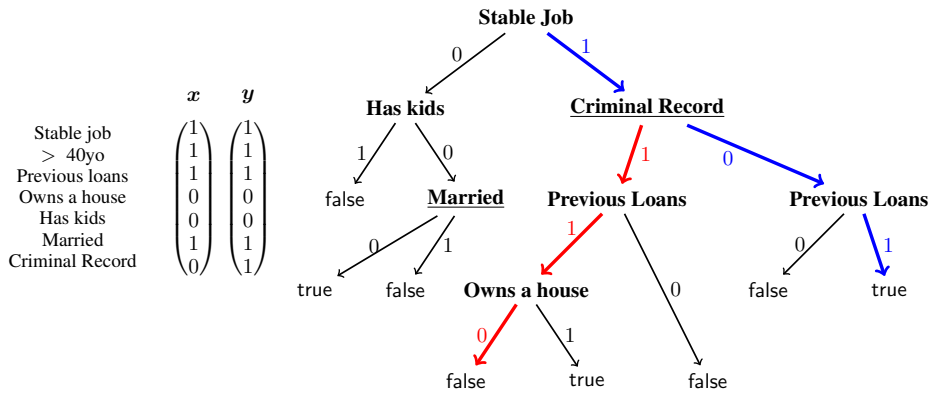


Figure 2: Illustration of the bias in a decision tree used for judging loan applications. Instance $\boldsymbol{x}$ follows the blue path, while instance $\boldsymbol{y}$ follows the red path. Protected features are underlined.

we focus on a (very) restricted and simplistic concept of bias and start by considering a simplistic approach to fairness; *fairness through unawareness* [16, 24]. In this notion, a certain subset of the features (that is, components of the input) is said to be *protected*, and consist of features that intuitively should not be used for decision taking. This usually includes features like gender, age, marital status, etc.

Even though this property is desirable, it requires the design of models to be originally conscious of what features to take into account, and which features to consider protected. As the concept of *protected* is a dynamic one, that might not always be possible. This motivates the need for being able to, given a decision and a set of features, decide whether those features were relevant in the decision.

We take the formalization of Darwiche and Hirth [11] of a more flexible notion than *not explictly used*, which is to say that the protected features are not being decisive for any input instance. Note that this notion strictly implies other weaker notions as that of *Demographic Parity* [24], in which one expects members of different protected groups to have the same probability of being rejected or accepted by the considered model.

Definition 3 (Biased decision and model)**.** *Given a model $\mathcal{M}$, of input size n, and a set of protected features $\mathcal{P} \subseteq \{1, \ldots, n\}$, an instance $\boldsymbol{x}$ is said to be a biased decision of $\mathcal{M}$ if and only if there exists an instance $\boldsymbol{y}$ such that $\boldsymbol{x}$ and $\boldsymbol{y}$ differ only on features in $\mathcal{P}$ and $\mathcal{M}(\boldsymbol{x}) \neq \mathcal{M}(\boldsymbol{y})$.*

A model $\mathcal{M}$ is said to be biased iff there is an instance $\boldsymbol{x}$ that is a biased decision of $\mathcal{M}$. □

Example 1. Consider that among the features used by a bank to judge loan applications we find both the *criminal record* and the *marital status* of applicants. If the law forbids said considerations, deeming the features *Criminal Record* and *Married* to be protected, then individuals could rebut decisions they received that could have unfairly been based on such features. Moreover, one would be interesting in analyzing a posteriori, given a trained model, whether there is actually an individual who could have been classified unfairly. As illustrated in Figure 2, this particular FBDD is biased, as it presents a biased decision. The instance $\boldsymbol{x}$ is a positive instance of the model, while the negative instance $\boldsymbol{y}$ differs from it only on the protected feature corresponding to the criminal record. □

We now show how to encode these queries in the proposed logical structure, assuming the set of protected features $\mathcal{P}$ is fixed. For simplicity in what follows we use $\mathcal{P}^c$ to denote all non-protected features.

First, we use the constant symbols $\mathbf{0}^i$ (respectively $\mathbf{1}^i$) to refer to the vectors that have 0 (resp. 1) in the i-th component and $\bot$ everywhere else. With this we define the following auxiliary formulas for every integer $1 \leq i \leq n$ and set S of integers in $\{1, \ldots, n\}$:

$$\textsc{Match}_i(\boldsymbol{x}, \boldsymbol{y}) \equiv (\mathbf{0}^i \subseteq \boldsymbol{x} \wedge \mathbf{0}^i \subseteq \boldsymbol{y}) \vee (\mathbf{1}^i \subseteq \boldsymbol{x} \wedge \mathbf{1}^i \subseteq \boldsymbol{y}) \tag{7}$$

$$\textsc{Match}_S(\boldsymbol{x}, \boldsymbol{y}) \equiv \bigwedge_{i \in S} \textsc{Match}_i(\boldsymbol{x}, \boldsymbol{y}) \tag{8}$$

The first formula says that instances $\boldsymbol{x}$ and $\boldsymbol{y}$ have the same value in the i-th component, while the second one states that they have matching values for each component in the set S.

$$\begin{aligned} \textsc{BiasedDecision}(\boldsymbol{x}) &\equiv \textsc{Full}(\boldsymbol{x}) \wedge \exists \boldsymbol{y}(\textsc{Full}(\boldsymbol{y}) \wedge \neg \textsc{SameClass}(\boldsymbol{x}, \boldsymbol{y}) \wedge \textsc{Match}_{\mathcal{P}^c}(\boldsymbol{x}, \boldsymbol{y})) \\ \textsc{BiasedModel} &\equiv \exists \boldsymbol{x} \textsc{BiasedDecision}(\boldsymbol{x}) \end{aligned}$$

The first formula states that $\boldsymbol{x}$ is an instance, and that there is an instance $\boldsymbol{y}$ with the opposite classification such that they match on every non-protected feature, and thus differ only on protected features.

4 Complexity Results

We begin by briefly introducing the classes of models under which our results apply. We consider the class of perceptrons and Multilayer perceptrons (MLP) with ReLU activations (for the sake of space we relegate their precise definition for the appendix). We also consider a generalization of decision trees in the form of *free binary decision diagrams* (FBDDs) that we next formalize. A *binary decision diagram* (BDD [36]) is a rooted directed acyclic graph $\mathcal{M}$ with labels on edges and nodes, where (i) each leaf is labeled with true or with false; (ii) each internal node (a node that is not a leaf) is labeled with an element of $\{1, \ldots, n\}$; and (iii) each internal node has an outgoing edge labeled 1 and another one labeled 0. Instances are classified according to the path they describe from the root to a leaf, in the same way a decision tree does. A binary decision diagram $\mathcal{M}$ is *free* (FBDD) if for every path from the root to a leaf, no two nodes on that path have the same label. Note that a *decision tree* is a particular case of an FBDD, in which the underlying graph is a tree.

There are several ways to study the complexity of answering queries over data (a model in our case), depending on whether the size of the query, the data, or both, are fixed [21]. In order to present a brief discussion, we will study the problem of evaluating queries depending on bounds on their size. As shown in Section 3, while certain queries like *minimal sufficient reason* have constant size, some others like *biased decision* have size that it is up to linear with respect to the model size. We study this dependency by considering a family of problems parameterized by a function f, which determines the relative length of queries with respect to the underlying model. This is reminiscent of *paramaterized complexity* [13, 15], in which problems are studied by differentiating between the instance size and a parameter, which is in general much smaller than the instance size.

Problem:	EVALUATION-f
Input:	A model $\mathcal{M}$, that induces a structure $\mathfrak{A}$, a formula φ over σ such that $\lvert\varphi\rvert \leq f(\lvert\mathcal{M}\rvert)$.
Output:	YES, if $\mathfrak{A} \models \varphi$ and NO otherwise

As mentioned before, the structure $\mathfrak{A}$ is always implicit in the input, as its size can be exponential with respect to the model $\mathcal{M}$. In a slight abuse of notation, we present results using classes of functions instead of particular functions for the EVALUATION problem. That is, we describe the complexity of problems of the form EVALUATION-C, for C a class of functions. When doing so for a complexity upper bound, we mean that the upper bound holds for every function $f \in C$. When doing so for a lower bound, we mean that there exists a function $f \in C$ for which the bound holds.

The following theorem, whose proof follows by standard results on the complexity of evaluating FO formulas, sets an upper-bound for the complexity of evaluating polynomially-long queries.

Theorem 1. *The* EVALUATION-*poly*(n) *problem is in* PSPACE *for any model $\mathcal{M}$ such that $\mathcal{M}(\boldsymbol{x})$ is polynomial space computable for every $\boldsymbol{x}$.*

Next theorem states that, for MLPs, even constant-size queries are hard to evaluate.

Theorem 2. *The* EVALUATION-$O(1)$ *problem is* NP*-hard and* co-NP*-hard for MLPs, even over formulas with only one quantifier.*

The proof of the previous theorem follows from the simple observation that, given a boolean formula φ, one can compute an MLP $\mathcal{M}_\varphi$ that is equivalent to φ (as boolean functions) in polynomial time [6]. Then, by building the structure $\mathfrak{A}$ over $\mathcal{M}_\varphi$, the queries $\exists \boldsymbol{x} \textsc{Positive}(\boldsymbol{x})$ and $\forall \boldsymbol{x} \textsc{Positive}(\boldsymbol{x})$ encode the standard problems SAT and TAUT [3], respectively, from which hardness for both classes is derived.

Note that the previous theorem exploits only one quantifier, and does not use the $\subseteq$ predicate at all. It is easy to see that the more general case of evaluating a constant size formula belongs to the class $\text{BH} = \text{QH} = P^{\text{NP}[O(1)]}$ [34], of problems solvable by a polynomial time DTM with a constant number of queries to an NP oracle, and we conjecture that for every fixed $k \geq 1$, there are formulas of size $O(k)$ over $\mathfrak{A}$ which are hard for QH_k [34], the classes defined by allowing a polynomial time DTM to make k queries to an NP oracle. A more detailed analysis remains as future work.

Theorem 3. *The* EVALUATION-$O(n)$ *problem is* PSPACE*-hard for FBDDs and perceptrons, and is already* NP*-hard even if the formula φ has only one quantifier.*

Proof sketch. The first part is proven by a reduction from TQBF, a standard PSPACE-hard problem [3]. The proof of the second part is more involved, and assumes nothing about the models, as it only uses the $\subseteq$ predicate. The key idea is to show that the linear size formulas using only the $\subseteq$ predicate can encode arbitrary instances of a restricted (but still NP-hard) variant of CNF-SAT in which every clause has either only positive literals or only negative literals. □

The following theorem states an upper bound for the difficulty of evaluating queries over models traditionally deemed interpretable, as FBDDs and perceptrons. Let us call existential (resp. universal) to formulas consisting in a sequence of existential (resp. universal) quantifiers followed by a quantifier-free formula.

Theorem 4. *The* EVALUATION-$O(1)$ *problem restricted to existential or universal formulas is in* PTIME *for FBDDs and perceptrons.*

Proof sketch. The proof is constructive, and the key element that forbids the theorem to hold over MLPs, is that both FBDDs and perceptrons allow us to efficiently determine, given a partial instance, whether none, some, or all its completions are positive. Such a check is NP-hard for MLPs. □

In a nutshell, our results distinguish the complexity of evaluating queries when the underlying model is an MLP (high complexity) from the case when the underlying model is an FBDD or a perceptron (lower complexity).

5 Towards tractability

While the previous section establishes theoretical intractability results, this section presents a more optimistic point of view by considering two directions that may allow for practical tractability: *interactive proofs* and *knowledge compilation*.

5.1 Interactive Proofs

A natural use-case for a language tailored for interpretability and bias detection is that of auditing models that are being used by companies or government institutions. However, it is often the case that the computing power of such institutions is greater than that of a regular citizen by several orders of magnitude. While this situation allows for entities with significant computing power to use models consisting of millions of features and parameters, which would make any reasoning about them intractable, it also gives a hint towards tractability. Consider a setting in which a user X wants to solve an interpretability query over a model published by a company Y. If the computational complexity of solving said query is in the class NP, even if X may not have the computing power to compute it, it could be that company Y does. Then Y can send a short certificate of its answers to X (because of the membership in NP) which X can then quickly verify. As discussed in Section 4, some of the considered queries are co-NP-hard, and even potentially PSPACE-hard. This implies, under standard theoretical complexity assumptions (such as NP $\neq$ co-NP [3]), that there are no polynomially-short certificates for these problems. *Interactive Proofs* are a potential solution to this shortcoming. Interactive Proofs are abstract protocols in which two parties, a prover and a verifier, exchange messages, by rounds, with a particular goal: for the prover to prove something to the verifier. The complexity class IP [3] contains all problems for which a solution can be proved to a polynomial-time verifier in polynomially many rounds, up to arbitrarily high probability. The fact that IP = PSPACE [22, 29] guarantees that all the previously mentioned problems can be treated in this fashion. That is, entities with large computational power, or distributed computing systems, can

prove to individuals with less computational power that their queries are being answered faithfully up to an arbitrarily high degree of confidence.

5.2 Knowledge Compilation

As shown in Section 4, there is a significant difference in complexity for evaluating queries over models traditionally deemed interpretable (e.g., decision trees) versus models traditionally deemed opaque (e.g., MLPs). This difference in complexity applies for every query made in a session by a user, meaning for example that if a user is to make 10 queries to interpret a deep MLP, each of those 10 queries could require a large amount of computational resources. Such scenario naturally suggests that it would be convenient to first transform an MLP to a decision tree (even if for a high computational cost) in order for posterior queries to run faster. This is precisely the benefit of *Knowledge Compilation* [12]. Several authors [30, 31, 32] have explored this avenue with increasing success, providing algorithms that, while exponential in the worst-case, have allowed for efficiently compiling small binary neural networks into succinct decision diagrams [30].

6 Future work: *logical extensions*

While the minimalistic structure presented in Section 2 can already express several different interpretability queries, it is still arguably limited in terms of its expressive power. Our future work aims to study extensions to the logic that allow both for expressing new queries, and reducing the size of queries that were expressible before. We briefly discuss some natural extensions.

Counting The semantics and complexity of adding counting operators to First Order Logic have been studied in depth [2, 14], and would allow in our case to express properties related to the proportion of instances that satisfy certain properties. For example, one could ask whether the number of accepted instances for the bank loan problem in which the person has kids is higher or lower than for people without kids.

Model comparison A natural need one may face is that of interpreting a model, not on its own, but in comparison to a previously deployed model. For example, queries like "*is there an instance of the bank loan problem accepted by $\mathcal{M}_1$ and rejected by $\mathcal{M}_2$ that is married?*" could be expressed by considering, instead of a single predicate POSITIVE, a sequence of predicates $\text{POSITIVE}_1, \ldots, \text{POSITIVE}_\ell$, where each predicate would naturally be interpreted according to a sequence of models $\mathcal{M}_1, \ldots, \mathcal{M}_\ell$. It is easy to see as well that this extension allows for querying model equivalence, and paired with the previous extension it could allow for a more nuanced study of the difference of models, by counting for example on how many instances they differ.

Adding functions A binary function δ expressing the pointwise-difference between two instances can allow for expressing new queries and make certain queries more succinctly expressible. For example the $\text{BIASDECISION}(x)$ query can be simplified down to constant size by replacing $\text{MATCH}_{\mathcal{P}^c}(\boldsymbol{x}, \boldsymbol{y})$ by $\delta(\boldsymbol{x}, \boldsymbol{y}) \subseteq \vec{P}$, where $\vec{P}$ is the constant instance having 1 in every protected feature and 0 elsewhere. It is not hard to see that, if one also adds unary predicates HW_k, saying that the hamming weight of an instance is at most k $(0 \leq k \leq n)$, counterfactual interpretability queries like *Minimum Change Required*[1] (the minimum number of features to change required for changing the verdict of a given instance) [6] become succinctly expressible.

A dedicated study to the expressiveness and complexity of such extensions is part of our future work, and presents some interesting theoretical challenges. For example, considering the second extension, one can write a constant size query stating that two input models are equivalent[2]. It is known that evaluating equivalence between perceptrons is NP-hard [30], and it is not currently known whether it can be done in polynomial time for FBDDs [12], although a randomized polynomial time algorithm is known [8].

[1] also called *Robustness* [11].

[2] $\forall \boldsymbol{x}(\text{POSITIVE}_1(\boldsymbol{x}) \leftrightarrow \text{POSITIVE}_2(\boldsymbol{x}))$

Broader Impact

Although interpretability as a subject may have a broad practical impact, our results in this paper are mostly theoretic, so we think that this work does not present any foreseeable societal consequences.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] M. Arenas, M. Muñoz, and C. Riveros. Descriptive complexity for counting complexity classes. *Logical Methods in Computer Science ; Volume 16*, pages Issue 1 ; 1860–5974, 2020.

[3] S. Arora and B. Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[4] A. B. Arrieta, N. Díaz-Rodríguez, J. D. Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera. Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58:82–115, 2020.

[5] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[6] P. Barceló, M. Monet, J. Pérez, and B. Subercaseaux. Model interpretability through the lens of computational complexity. 2020.

[7] V. Belle. Logic meets probability: Towards explainable AI systems for uncertain worlds. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, Aug. 2017.

[8] M. Blum, A. K. Chandra, and M. N. Wegman. Equivalence of Free Boolean Graphs can be Decided Probabilistically in Polynomial Time. *Inf. Process. Lett.*, 10:80–82, 1980.

[9] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. MIT Press, 2001.

[10] A. Darwiche. Three modern roles for logic in AI. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. ACM, May 2020.

[11] A. Darwiche and A. Hirth. On the reasons behind decisions. *arXiv preprint arXiv:2002.09284*, 2020.

[12] A. Darwiche and P. Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, Sept. 2002.

[13] R. G. Downey and M. R. Fellows. *Parameterized complexity*. Monographs in Computer Science. Springer, 1999.

[14] M. Droste and P. Gastin. Weighted automata and weighted logics. In L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *Automata, Languages and Programming*, pages 513–525, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[15] J. Flum and M. Grohe. *Parameterized complexity theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[16] P. Gajane and M. Pechenizkiy. On Formalizing Fairness in Prediction with Machine Learning, 2017.

[17] J. Goldsmith, M. Hagen, and M. Mundhenk. Complexity of DNF minimization and isomorphism testing for monotone formulas. *Information and Computation*, 206(6):760–775, 2008.

[18] S. R. Hong, J. Hullman, and E. Bertini. Human factors in model interpretability: Industry practices, challenges, and needs. *Proceedings of the ACM on Human-Computer Interaction*, 4(CSCW1):1–26, May 2020.

[19] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, USA, 2004.

[20] H. Kaur, H. Nori, S. Jenkins, R. Caruana, H. Wallach, and J. W. Vaughan. Interpreting interpretability: Understanding data scientists' use of interpretability tools for machine learning. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM, Apr. 2020.

[21] L. Libkin. *Elements of Finite Model Theory*. Springer Berlin Heidelberg, 2004.

[22] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, Oct. 1992.

[23] S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017.

[24] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan. A Survey on Bias and Fairness in Machine Learning, 2019.

[25] E. Ntoutsi, P. Fafalios, U. Gadiraju, V. Iosifidis, W. Nejdl, M.-E. Vidal, S. Ruggieri, F. Turini, S. Papadopoulos, E. Krasanakis, I. Kompatsiaris, K. Kinder-Kurlanda, C. Wagner, F. Karimi, M. Fernandez, H. Alani, B. Berendt, T. Kruegel, C. Heinze, K. Broelemann, G. Kasneci, T. Tiropanis, and S. Staab. Bias in data-driven artificial intelligence systems—An introductory survey. *WIREs Data Mining and Knowledge Discovery*, 10(3), Feb. 2020.

[26] M. T. Ribeiro, S. Singh, and C. Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144, 2016.

[27] C. Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

[28] P. Saleiro, B. Kuester, L. Hinkson, J. London, A. Stevens, A. Anisfeld, K. T. Rodolfa, and R. Ghani. Aequitas: A bias and fairness audit toolkit, 2019.

[29] A. Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, Oct. 1992.

[30] W. Shi, A. Shih, A. Darwiche, and A. Choi. On tractable representations of binary neural networks. *arXiv preprint arXiv:2004.02082*, 2020.

[31] A. Shih, A. Choi, and A. Darwiche. A symbolic approach to explaining Bayesian network classifiers. *arXiv preprint arXiv:1805.03364*, 2018.

[32] A. Shih, A. Darwiche, and A. Choi. Verifying binarized neural networks by Angluin-style learning. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 354–370. Springer, 2019.

[33] C. Umans. The minimum equivalent DNF problem and shortest implicants. *Journal of Computer and System Sciences*, 63(4):597–611, 2001.

[34] K. W. Wagner. Bounded query classes. *SIAM Journal on Computing*, 19(5):833–846, Oct. 1990.

[35] D. Watson. Conceptual challenges for interpretable machine learning. *SSRN Electronic Journal*, 2020.

[36] I. Wegener. BDDs—design, analysis, complexity, and applications. *Discrete Applied Mathematics*, 138(1-2):229–251, 2004.

In order to simplify our proofs we make a standard syntactical assumption; that all formulas are given in *prenex* normal form, that is, with all quantifiers at the beginning and negations appearing only directly in front of terms. It is well known that every formula in first order logic can be rewritten in prenex form, taking only polynomial time to do so [21].

As an example, the query SUFFICIENTREASON discussed in Section 3, presented as a formula with two free variables, can be expressed in prenex form as

$$\psi(\boldsymbol{x},\boldsymbol{y}) \equiv \forall\boldsymbol{v}\forall\boldsymbol{w}\exists\boldsymbol{z}\Big[(\boldsymbol{y}\subseteq\boldsymbol{x})\wedge\big(\neg(\boldsymbol{x}\subseteq\boldsymbol{v})\vee(\boldsymbol{x}=\boldsymbol{v})\big)\wedge \\ \big\{(\boldsymbol{w}\subseteq\boldsymbol{z}\wedge\neg(\boldsymbol{w}=\boldsymbol{z}))\vee\neg(\boldsymbol{y}\subseteq\boldsymbol{w})\vee(\mathrm{P}(\boldsymbol{w})\wedge\mathrm{P}(\boldsymbol{x}))\vee(\neg\mathrm{P}(\boldsymbol{w})\wedge\neg\mathrm{P}(\boldsymbol{x}))\big\}\Big]$$

Where the predicate POSITIVE has been abbreviated to P.

Before proving our results, w e state a simple lemma that will helps us along the way.

Lemma 1. *The only terms in formulas over $\mathfrak{A}$ are of the form (possibly negated)* POSITIVE$(\boldsymbol{x})$*, $\boldsymbol{y}\subseteq\boldsymbol{x}$ and $\boldsymbol{x}=\boldsymbol{y}$, and given a valuation of variables $\boldsymbol{x},\boldsymbol{y}$, each of these can be checked in polynomial time for FBDDs, perceptrons and MLPs.*

Proof. Direct from the fact that (partial) instances have polynomial size with respect to any model for them, and for each of the considered models it takes only polynomial time to determine whether a given instance is positive or not. □

Theorem 1. *The* EVALUATION-*poly*(n) *problem is in* PSPACE *for any model $\mathcal{M}$ such that $\mathcal{M}(\boldsymbol{x})$ is polynomial space computable for every $\boldsymbol{x}$.*

Proof. There are at most $|\mathcal{M}|^c$ free variables, for some constant c. We show that one can exhaustively try every possible valuation of them, and with that evaluate the whole formula in polynomial space. Indeed, as each variable can take at most $3^{|\mathcal{M}|}$ possible values, there at most $\left(3^{|\mathcal{M}|}\right)^{|\mathcal{M}|^c}=3^{|\mathcal{M}|^{c+1}}$ possible valuations. Therefore, we can represent valuations using no more than $\log 3^{|\mathcal{M}|^{c+1}}=O(|\mathcal{M}|^{c+1})$ bits. Based on Lemma 1, once a valuation is fixed, we can check whether it satisfies the formula using extra polynomial space. This yields a polynomial space algorithm. □

In order to prove Theorem 3, we prove each of its statements as a separate lemma.

Lemma 2. *The* EVALUATION-$O(n)$ *problem is* PSPACE*-hard for FBDDs and perceptrons.*

Proof. We show a reduction from the standard TQBF problem [3], in which one is given a quantified Boolean formula and asked to decide its truth value. Let $\chi=Q_1x_1Q_2x_2\cdots Q_nx_n\phi(x_1,\ldots,x_n)$ be an input instance of TQBF. First, it is easy to see that, both for FBDDs and perceptrons, one can build a model $\mathcal{M}$ of dimension n such that

$$\mathcal{M}(\boldsymbol{x})=\begin{cases}1 & \text{if } x_1=1\\ 0 & \text{otherwise}\end{cases}$$

Let us consider the structure $\mathfrak{A}$ induced by $\mathcal{M}$. Then let ψ be a formula (with free variables) over $\mathfrak{A}$, syntactically constructed from ϕ, by replacing every occurrence of a variable x_i in ϕ by the term POSITIVE$(\boldsymbol{x}_i)$ (which we will abbreviate as $P(\boldsymbol{x}_i)$ henceforth) where the variables $\boldsymbol{x}_i$[3] are the free variables of ψ. Now consider the following formula over $\mathfrak{A}$:

$$\xi=Q_1\boldsymbol{x}_1Q_2\boldsymbol{x}_2\cdots Q_n\boldsymbol{x}_n\psi(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n)$$

We claim that $\mathfrak{A}\models\xi$ if and only if $\chi\in$ TQBF. Indeed, as the terms $P(\boldsymbol{x}_i)$ can effectively take both values true and false, depending on the particular instance $\boldsymbol{x}_i$ that is bound, we can see equisatisfiability (and thus finish the proof) by doing induction over n, the number of variables.

[3]Note that $\boldsymbol{x}_i$ is the i-th of a sequence of variables over $\mathfrak{A}$, and is not to be confused with the i-th component of a (partial) instance $\boldsymbol{x}$.

- **(Base case $n = 1$)** The formula χ is either of the form $\exists x_1 \phi(x_1)$ or $\forall x_1 \phi(x_1)$, where ϕ does not have any quantifiers and thus x_1 is the only variable we are dealing with. Let us focus on the case $\exists \phi(x_1)$ as the other case is analogous[4]. The semantics of FO indicate that $\exists x_1 \phi(x_1) \equiv \phi[x_1 \to 0] \vee \phi[x_1 \to 1]$, where $[A \to B]$ corresponds to replace every occurrence of the term A by the constant value B. In the considered case, $\xi \equiv \exists \boldsymbol{x}_1 \psi(\boldsymbol{x}_1)$ which has the same truth value as $\bigvee_{v \in \{0,1,\bot\}^1} \psi[\boldsymbol{x}_i \to v]$. Note that, by construction, $\boldsymbol{x}_i$ only appears in ψ on terms of the form $P(\boldsymbol{x}_i)$ and thus $\psi[\boldsymbol{x}_i \to v] \equiv \psi[P(\boldsymbol{x}_i) \to \mathcal{M}(v)]$. Recall that, by construction of ψ, it happens that $\psi[P(\boldsymbol{x}_1) \to \mathcal{M}(v)]$ has the same truth value as $\phi[x_1 \to v]$. As v takes values in $\{0, 1, \bot\}^1$, we have that $\mathcal{M}(v)$ effectively takes both values 0 and 1, and therefore we have that $\chi \equiv \bigvee_{v \in \{0,1\}} \phi[x_1 \to v]$ has the same truth value as $\bigvee_{v \in \{0,1\}} \psi[P(\boldsymbol{x}_1) \to \mathcal{M}(v)]$ which in turn has the same truth value of ξ. This concludes the base case.
- **(Inductive case)** The formula χ is either of the form $\exists x_1 \phi(x_1)$ or $\forall x_1 \phi(x_1)$, but where now $\phi(x_1)$ has all its $n - 1$ variables $x_2, \dots x_n$ as bound variables. Note that $\chi'(v) \equiv \phi[x_1 \to v]$ is a formula with $n - 1$ bound variables (and no other variables) so by our inductive hypothesis it happens that it has the same truth value as its corresponding formula $\xi'(v) \equiv \psi[P(\boldsymbol{x}_1) \to v]$ over $\mathfrak{A}$, for any value $v \in \{0, 1\}$. Finally, we can reason by cases on the initial quantifier of χ. Consider the case where $\chi \equiv \forall x_1 \phi(x_1)$ as the other case is analogous. Again, by semantics of FO, the truth value of χ is that of $\bigwedge_{v \in \{0,1\}} \phi[x_1 \to v] \equiv \bigwedge_{v \in \{0,1\}} \chi'(v)$. But as we show next this is the same truth value of ξ. Indeed, ξ corresponds in truth value, by the semantics of FO over $\mathfrak{A}$, to $\bigwedge_{v \in \{0,1,\bot\}^n} \xi'(v) \equiv \bigwedge_{v \in \{0,1,\bot\}^n} \psi[P(\boldsymbol{x}_1) \to \mathcal{M}(v)]$. But by definition $\mathcal{M}(v)$ is 1 if v has a 1 in its first component and 0 otherwise, so we conclude that the truth value of ξ is simply equal to $\bigwedge_{v \in \{0,1\}} \psi[P(\boldsymbol{x}_1) \to v]$, which is the same, by the inductive hypothesis mentioned earlier, to that of χ. This concludes the proof.

□

We actually prove a stronger statement than the one explicitly stated in Theorem 3.

Lemma 3. *The* EVALUATION-$O(n)$ *problem over any class of models $\mathcal{C}$ is already* NP*-hard for formulas with only one existential quantifier.*

Proof. We reduce from CNF-SAT with a linear number of clauses, which is well known to be NP-hard from standard reductions. Consider a formula $\varphi = \bigwedge_i C_i$, where clause $C_i = \bigvee_j \ell_{ij}$. From each clause C_i, we create two clauses $C'_{i,1}$ and $C'_{i,2}$, by introducing a new variable s_i, in the following way:

$$\bigvee_j \ell_{ij} \rightsquigarrow \underbrace{\left(s_i \vee \bigvee_{\ell_{ij} \text{ is positive}} \ell_{ij} \right)}_{C'_{i,1}} \wedge \underbrace{\left(\neg s_i \vee \bigvee_{\ell_{ij} \text{ is negative}} \ell_{ij} \right)}_{C'_{i,2}} \tag{9}$$

It is easy to see that the formula $\varphi' = \bigwedge_i \left(C'_{i,1} \wedge C'_{i,2} \right)$ is equisatisfiable with respect to φ. We say φ' is a formula in Clause-Monotone-CNF, as every clause is has either only negative literals, or only positive literals. The previous transformation is enough to show that deciding the satisfiability of a formula in Clause-Monotone-CNF is NP-hard. We now show that there is a reduction from this problem to the EVALUATION-$O(n)$ problem for formulas with 1 existencial quantifier.

Indeed, consider a Clause-Monotone-CNF formula $\psi = \psi_P \wedge \psi_N$, where ψ_P is a CNF formula with a linear number of clauses in which all literals are positive, and ψ_N is a CNF formula with a linear number of clauses in which all literals are negative. For every clause C_i^P in ψ_P, we create a term $T_i^P \equiv \neg(\boldsymbol{x} \subseteq \boldsymbol{b}_i^P)$, whereas for every clause C_i^N in ψ_N, we create a term $T_i^N \equiv \neg(\boldsymbol{b}_i^N \subseteq \boldsymbol{x})$. Then, consider the formula ϕ which is simply an existencial search on the conjunction of these terms.

$$\phi \equiv \exists \boldsymbol{x} \left(T_1^N \wedge \cdots \wedge T_k^N \wedge T_1^P \wedge \cdots \wedge T_{k'}^P \right)$$

[4]We consider the case with a universal quantifier in the inductive case, to give the reader a complete analysis while avoiding to focus on too many details.

Note that ϕ has size linear in the number of variables of ψ, which we can denote n. Now consider an arbitrary model $\mathcal{M}$ of class $\mathcal{C}$ with input size n, and evaluate each free variable $\boldsymbol{b}$ of ϕ in the way we show next. If clause C_i^P includes variables with indices in a set S^i, let $\boldsymbol{b}_i^P$ be the partial instance that has $\perp$ in every component $j \in S^i$ and 1 elsewhere. Conversely, if clause $\boldsymbol{b}_i^N$, includes variables with indices in a set S^i, let $\boldsymbol{b}_i^N$ be the partial instance that has 1 in every component $j \in S^i$ and $\perp$ elsewhere. The following claim materializes our reduction.

Claim 1. *There is a partial instance $\boldsymbol{x}$ satisfying ϕ if and only if ψ is satisfiable.*

Proof of Claim 1. First, note that by construction any instance $\boldsymbol{x}$ has as many components as ψ has variables, so we can naturally associated the i-th component of $\boldsymbol{x}$ (x_i) to variable x_i in ψ. For the forward direction, consider a partial instance $\boldsymbol{x}$ that satisfies ϕ, and let σ be the valuation of variables of ψ that assigns true to variable x_i if and only if the instance $\boldsymbol{x}$ has 1 in its i-th component. We will show that σ satisfies every clause in ψ, and thus the entire formula. Consider an arbitrary positive clause $C_i^P \in \psi$. As $\boldsymbol{x}$ and $\boldsymbol{b}_i^P$ can only have $\perp$ or 1 in any component, by construction, and $\boldsymbol{x}$ satisfies the restriction $\neg(\boldsymbol{x} \subseteq \boldsymbol{b}_i^P)$, we know that there is at least one component j for which $\boldsymbol{x}$ has a 1 and $\boldsymbol{b}_i^P$ has $\perp$. But $\boldsymbol{b}_i^P$ has $\perp$ precisely in the components whose associated variables appear in the clause C_i^P, which means that $\boldsymbol{x}$ has a 1 in at least one of those, implying that σ assigns true to at least one of those, and thus satisfies clause C_i^P. Consider now an arbitrary negative clause $C_i^N \in \psi$. As $\boldsymbol{x}$ and $\boldsymbol{b}_i^N$ can only have $\perp$ or 1 in any component, by construction, and $\boldsymbol{x}$ satisfies the restriction $\neg(\boldsymbol{b}_i^N \subseteq \boldsymbol{x})$, we know that there is at least one component j for which $\boldsymbol{b}_i^N$ has a 1 while $\boldsymbol{x}$ has $\perp$. But $\boldsymbol{b}_i^N$ has a 1 precisely in the components whose associated variables appear in the clause C_i^N, which means that $\boldsymbol{x}$ has $\perp$ in at least one of those, implying that σ assigns false to at least one of those, and thus satisfies clause C_i^P. As all clauses C_i^N, C_i^P are satisfied by σ, the formula ψ is satisfied by σ.

For the backward direction, consider an assignment σ that satisfies ψ and thus every clause C_i^N, C_i^P. Let $\boldsymbol{x}$ be the partial instance that has 1 in every component associated with a variable that has been assigned true by σ, and $\perp$ elsewhere. Let us check that $\boldsymbol{x}$ satisfies every term. For a term of the form $T_i^P \equiv \neg(\boldsymbol{x} \subseteq \boldsymbol{b}_i^P)$, we have that $\boldsymbol{b}_i^P$ has $\perp$ precisely in the components associated with variables that appear in clause C_i^P, and as at least one of those is assigned true by σ, we have that $\boldsymbol{x}$ has a 1 in at least one of the components for which $\boldsymbol{b}_i^P$ has $\perp$, which is enough to satisfy the term. For a term of the form $T_i^N \equiv \neg(\boldsymbol{b}_i^N \subseteq \boldsymbol{x})$, we have that $\boldsymbol{b}_i^N$ has a 1 precisely in the components associated with variables that appear negated in clause C_i^N, and at least one of those variables is assigned false by σ, we have that $\boldsymbol{x}$ has $\perp$ in at least one of the components for which $\boldsymbol{b}_i^N$ has 1, which is enough to satisfy the term. As every term is satisfied, we have that the partial instance $\boldsymbol{x}$ satisfies ϕ. □

As the reduction holds regardless of the considered class of models, it works for all of them.

□

We are now ready to prove Theorem 3.

Theorem 3. *The* EVALUATION-$O(n)$ *problem is* PSPACE*-hard for FBDDs and perceptrons, and is already* NP*-hard even if the formula φ has only one quantifier.*

Proof. Follows trivially by combining Lemma 2 and Lemma 3. □

In order to make the proof of Theorem 4 easier, we start by proving the following lemma.

Lemma 4. *Given an FBDD or perceptron $\mathcal{M}$ and a partial instance $\boldsymbol{y}$ one can decide in polynomial time whether there is a positive completion of $\boldsymbol{y}$ under $\mathcal{M}$.*

Proof. The case for FBDDs is easy, as it follows directly from the fact that one can count the number of positive completions over a partial instance $\boldsymbol{y}$ in polynomial time for an FBDD [6]. Indeed, if we can count the number of positive completions of $\boldsymbol{y}$, then it is enough to check whether that number is greater than 0. The case for perceptrons follows from the fact that, given a partial instance $\boldsymbol{y}$ and an instance $\boldsymbol{x}$, such that $\boldsymbol{y} \subseteq \boldsymbol{x}$, one can check in polynomial time whether all completions of $\boldsymbol{y}$ have the same classification as $\boldsymbol{x}$ [6]. Indeed, consider any completion $\boldsymbol{x}$ of $\boldsymbol{y}$, for example filling with 0 the undefined components. If $\mathcal{M}(\boldsymbol{x}) = 1$, then we are done. Otherwise, $\mathcal{M}(\boldsymbol{x}) = 0$ and it is enough to

use the previous fact, as if for all completions $\boldsymbol{z}$ of $\boldsymbol{y}$ it happens that $\mathcal{M}(\boldsymbol{z}) = \mathcal{M}(\boldsymbol{x}) = 0$ we simply reject the instance. □

Theorem 4. *The* EVALUATION-$O(1)$ *problem, restricted to existential formulas or universal formulas is in* PTIME *for FBDDs and perceptrons.*

Proof. We denote by $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_c$ the constants in the given formula, by $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$ the bound variables, and by d the dimension of the input model.

Let us quickly discard the case in which the input formula does not have any quantifiers, as using Lemma 1 is enough to solve the problem. Now, we focus on the case where the input formula contains only existential quantifiers. This will be enough for the whole theorem, as if the input where to be a universal formula, we could evaluate its negation, which is existential, and then negate the result.

We start by reducing every constant term (that is, every term that does not contain a bound variable) to its truth value computed by using Lemma 1 and reducing constants according to the standard rules over the Boolean connectives. As a result, every term contains a bound variable. Note that every equality of variables can be rewritten using $\subseteq$ as a double inclusion, which allows to simplify our analysis to the following eight *elemental terms*:

1. $\textsc{Positive}(\boldsymbol{x}_i)$
2. $\neg\textsc{Positive}(\boldsymbol{x}_i)$
3. $\boldsymbol{b}_j \subseteq \boldsymbol{x}_i$
4. $\neg(\boldsymbol{b}_j \subseteq \boldsymbol{x}_i)$
5. $\boldsymbol{x}_i \subseteq \boldsymbol{b}_j$
6. $\neg(\boldsymbol{x}_i \subseteq \boldsymbol{b}_j)$
7. $\boldsymbol{x}_i \subseteq \boldsymbol{x}_k$
8. $\neg(\boldsymbol{x}_i \subseteq \boldsymbol{x}_k)$

Let us call *term assignment* to a function that assigns every elemental term to either true or false. Note that for an existential formula to be true, there must be a term assignment τ^* that satisfies the whole formula and such that there is an assignment of the quantified variables that makes every elemental term evaluate to the same value that τ^* dictates.

As the input formula has a constant number of elemental terms, we can afford to exhaustively try every term assignments. Note that, after the truth value of every elemental term is fixed, the whole value of the formula can be obtained by simply reducing connectives. This allows to create a list Γ of term assignments that make the whole formula true, which is of polynomial size. This reduces our problem to, given a term assignment $\tau \in \Gamma$, decide whether there is an assignment of the quantified variables that makes every term take the truth value dictated by τ. We will say such an assignment of the quantified variables is *compatible* with τ. If for some $\tau \in \Gamma$ we find a compatible assignment of the quantified variables, then the input formula is true, and otherwise we can confidently reject it.

Let us illustrate the ideas so far with a running example.

Example 2. *Consider $n = 5$, and an FBDD $\mathcal{M}$ that accepts inputs with a least 3 features having a value of 1. For instance $(1 \quad 0 \quad 1 \quad 0 \quad 1)$ is accepted by $\mathcal{M}$, while $(0 \quad 0 \quad 1 \quad 1 \quad 0)$ is not. Then, let input (partial) instances be $\boldsymbol{b}_1 = (1 \quad 1 \quad 1 \quad 1 \quad 0)$ and $\boldsymbol{b}_2 = (\bot \quad 1 \quad \bot \quad 1 \quad \bot)$, and consider the following input formula:*

$$\exists \boldsymbol{x}_1 \Big(\textsc{Positive}(\boldsymbol{x}_1) \wedge \textsc{Positive}(\boldsymbol{b}_1) \wedge (\boldsymbol{x}_1 \subseteq \boldsymbol{b}_1 \vee \boldsymbol{b}_2 \subseteq \boldsymbol{x}) \wedge \neg(\boldsymbol{x}_1 = \boldsymbol{b}_1) \Big)$$

which we simplify and rewrite as

$$\exists \boldsymbol{x}_1 \Big(\textsc{Positive}(\boldsymbol{x}_1) \wedge (\boldsymbol{x}_1 \subseteq \boldsymbol{b}_1 \vee \boldsymbol{b}_2 \subseteq \boldsymbol{x}_1) \wedge \big(\neg(\boldsymbol{x}_1 \subseteq \boldsymbol{b}_1) \vee \neg(\boldsymbol{b}_1 \subseteq \boldsymbol{x}_1) \big) \Big)$$

Which, is of the form

$$\exists \boldsymbol{x}_1 \Big(\Theta_1 \wedge (\Theta_2 \vee \Theta_3) \wedge \big(\Theta_4 \vee \Theta_5\big) \Big)$$

where the Θ_i are elemental terms.

We will now see how to check whether a given term assignment τ has a compatible assignment of the quantified variables. Once an assignment for the terms is fixed, what we have is a set of *restrictions*, and we wonder whether there is a (partial) instance that satisfies all restrictions simultaneously.

Let us discuss how to handle restrictions of the form 3-6, forgetting by now about 1-2. We will check whether there are instances that satisfy such restrictions with the aid of propositional logic. Let us define variables $0^i_j, 1^i_j, \bot^i_j$ for $1 \leq j \leq m$ and $1 \leq i \leq d$. Where 0^i_j means that the j-th feature of the i-th quantified variable has value 0, and $1^i_j, \bot^i_j$ are defined analogously. Note immediately that, as each component can have at most one value, we want that for every pair (i, j), exactly one variable out of $0^i_j, 1^i_j, \bot^i_j$ is assigned true. We now show how to express restrictions $3 - 6$ as propositional formulas over these variables. A restriction of the form $\boldsymbol{x}_i \subseteq \boldsymbol{b}$ means that every defined component of $\boldsymbol{b}$ is either undefined in $\boldsymbol{x}_i$ or defined to its same value. For example, if $\boldsymbol{b}_7 = 1$, then $\bot^i_7 \vee 1^i_7$, or equivalently $\neg 0^i_7$. We will keep the latter representation. On the other hand, every undefined component of $\boldsymbol{b}$ must be undefined as well in $\boldsymbol{x}_i$. Therefore, we have the following rewriting rules[5]:

$$(\boldsymbol{x}_i \subseteq \boldsymbol{b}) \rightsquigarrow \ell_1 \wedge \cdots \wedge \ell_d \text{ where } \ell_j \equiv \bot^i_j \text{ if } b_j = \bot \text{ and } \ell_j \equiv \neg(1 - b_j)^i_j \text{ otherwise.} \tag{10}$$

Similarly, we have the following rules as well

$$(\boldsymbol{b} \subseteq \boldsymbol{x}_i) \rightsquigarrow \ell_k \wedge \cdots \wedge \ell_{k'} \text{ where } \ell_j \equiv (b_j)^i_j \text{ for every } j \text{ such that } b_j \neq \bot. \tag{11}$$

Note that the last equation does not necessarily introduce d literals, as the positions for which $\boldsymbol{b}_i$ is equal to $\bot$ impose no restrictions on $\boldsymbol{x}$. Analogously, we pose:

$$\neg(\boldsymbol{x}_i \subseteq \boldsymbol{b}) \rightsquigarrow \ell_1 \vee \cdots \vee \ell_d \text{ where } \ell_j \equiv \neg\bot^i_j \text{ if } b_j = \bot \text{ and } \ell_j \equiv (1 - b_j)^i_j \text{ otherwise.} \tag{12}$$

as well as

$$\neg(\boldsymbol{b} \subseteq \boldsymbol{x}_i) \rightsquigarrow \ell_k \vee \cdots \vee \ell_{k'} \text{ where } \ell_j \equiv \neg(b_j)^i_j \text{ for every } j \text{ such that } b_j \neq \bot. \tag{13}$$

We call formulas on the right hand side of Equations 10 and 11 *conjunctives*, whereas those in Equations 12 and 13 will be called *disjunctives*.

Conjunctive formulas are easy to handle, as they immediately prescribe the values of certain variables $0^i_j, 1^i_j, \bot^i_j$. So we can process each of those formulas, and reject immediately if we find a contradiction. Otherwise, we proceed to determine, having fixed certain variables according to the processing of every conjunctive formula, whether there is an assignment of variables that satisfies every disjunctive formula.

As there is a constant number of elemental terms, there is also a constant number of disjunctive formulas to satisfy. We can therefore afford to guess, for each disjunctive formula, a literal that satisfies it. After guessing a set of literals that satisfies every formula, it is easy to check in polynomial time, for each literal in the set, whether it can be removed while keeping all formulas satisfied. This allows to obtain, in polynomial time, a list Λ of minimal sets of literals that satisfy every disjunctive formula. For each set in Λ, we need to check that it does not contradict the values obtained in the previous step when processing conjunctive formulas, and also that it does not directly break any restriction of the form 7 or 8. The sets in Λ that fail either of the previous checks are discarded, and then by combining each surviving set in Λ with the values prescribed by the processing of conjunctive formulas, we get a list Λ^* of *candidate assignments*. It remains to be seen whether it is possible to extend a candidate assignment so that it satisfies as well restrictions of the form 1 and 2, and that it still respects restrictions of the form 7 and 8, as we know already it will respect restrictions 3-6.

Let us illustrate the new ideas so far.

[5] In order to keep the notation clean, we consider a constant $\boldsymbol{b}$ with no index, but the same treatment holds for all constants $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_c$.

Example 3 (Continuation of Example 2)**.** *Maintaining our previous notation, we forget about* Θ_1 *and consider that* Θ_2 *is false,* Θ_3 *is true and* Θ_5 *is true. We omit* Θ_4*, assigned to* true*, as it simply the negation of* Θ_2*. Therefore we have the following restrictions:*

1. $\neg\Theta_2 \equiv \neg(\boldsymbol{x}_1 \subseteq \boldsymbol{b}_1) \rightsquigarrow 0_1^1 \vee 0_2^1 \vee 0_3^1 \vee 0_4^1 \vee 1_5^1$
2. $\Theta_3 \equiv (\boldsymbol{b}_2 \subseteq \boldsymbol{x}) \rightsquigarrow 1_2^1 \wedge 1_4^1$
3. $\Theta_5 \equiv \neg(\boldsymbol{b}_1 \subseteq \boldsymbol{x}) \rightsquigarrow \neg 1_1^1 \vee \neg 1_2^1 \vee \neg 1_3^1 \vee \neg 1_4^1 \vee \neg 0_5^1$

The second restriction imposes directly that the searched variable $\boldsymbol{x}_1$ *has a* 0 *on its second component, and a* 1 *on its fourth component. After applying the second restriction, the remaining two can be further simplified to:*

1. $0_1^1 \vee 0_3^1 \vee \vee 1_5^1$
2. $\neg 1_1^1 \vee 0_3^1 \vee \neg 0_5^1$

Then, the assignment $(0_1^1, \neg 1_1^1)$ *is viable, and can be simplified to* 0_1^1*. Together with the second restriction obtained in the first paragraph, we note that the assignment imposing* 0_1^1*,* 1_2^1*, and* 1_4^1 *is a candidate assignment. One can then check that, any instance respecting this candidate assignment, like* $(0 \quad 1 \quad 0 \quad 0 \quad 1)$ *respects the considered assignment for every term.*

As a consequence of the precedent steps, we have either found at least one candidate assignment for the variables that satisfies τ (except for the elemental terms of form 1-2), or that no such assignment exists. If no candidate assignment exists, we can safely discard τ as a term assignment and proceed to consider the next one. If we have found candidate assignments, we have to check whether there is any of them which can be extended to satisfy as well the restrictions of the form 1-2 and 7-8.

In order to consider restrictions of the form 7, we can first imagine that every restriction of the form $(\boldsymbol{x}_i \subseteq \boldsymbol{x}_k)$ is an edge between a node $\boldsymbol{x}_i$ and $\boldsymbol{x}_k$ in a directed graph. Strongly connected components in such graph impose that the involve variables must be equal, and after being collapsed to a single node, one gets a directed acyclic graph for which a topological order can be computed in polynomial time. Then, we can iterate through the variables according to the topological order, and for each quantified variable $\boldsymbol{x}_i$ we are iterating over, all propositional variables of the form 0_j^i or 1_j^i that are forced to be true by the candidate assignment we are considering, must propagate to 0_j^k or 1_j^k (respectively) for every k such that the graph indicates that $\boldsymbol{x}_i \subseteq \boldsymbol{x}_k$ (even if transitively). This will enforce that everything assigned so far by the current candidate assignment is validating restrictions of the form 7 thus far.

For each of the constantly many restrictions of the form $\neg(\boldsymbol{x}_i \subseteq \boldsymbol{x}_k)$ we can afford to guess a component $1 \leq j \leq d$ such that $\boldsymbol{x}_i$ will have a defined value (namely 0 or 1) on its j-th component, and $\boldsymbol{x}_k$ will not have it. For each of these guesses (note that there is a polynomial number of them), we have fully enforced restrictions of the form 3-6 and 8. Therefore it only remains to see whether we can find values for every component we have not assigned to anything yet such that restrictions of the form 1 and 2 are satisfied (while obviously not breaking restrictions of the form 7, which we have not fully enforced).

Let us say that a variable $\boldsymbol{x}_i$ for which we have the restriction $\textsc{Positive}(\boldsymbol{x}_i)$ is a *full variable*, as we need to give a defined value to each of its components. Every variable that is not full can be called a *partial variable*. Note that if we have a restriction of the form $(\boldsymbol{x}_i \subseteq \boldsymbol{x}_k)$, and $\boldsymbol{x}_i$ is a full variable, then it must hold that $\boldsymbol{x}_i = \boldsymbol{x}_k$, which simplifies our problem as one of them can be safely deleted, thus reducing the number of quantified variables. Therefore, when considering a restriction like $(\boldsymbol{x}_i \subseteq \boldsymbol{x}_k)$, we assume that $\boldsymbol{x}_i$ is not a full variable, and either $\boldsymbol{x}_k$ is a full variable, or it is not. If $\boldsymbol{x}_k$ is not a full variable, then nothing special needs to be done, as keeping $\boldsymbol{x}_k$ with undefined components (every component not forced to be 0 or 1 by the previous restrictions) will satisfy the restriction $(\boldsymbol{x}_i \subseteq \boldsymbol{x}_k)$ and also any potential restriction of the form $\neg\textsc{Positive}(\boldsymbol{x}_k)$. A border case is when the previous restriction have enforced every component in $\boldsymbol{x}_k$ to take either 0 or 1 and yet $\boldsymbol{x}_k$ is not a full instance. If that is the case, and we have a restriction of the form $\neg\textsc{Positive}(\boldsymbol{x}_k)$, then one simply checks whether that holds or not (if it holds then nothin needs to be done, but if it does not

then we discard the candidate assignment, as it not possible to satisfy said restriction. If, on the other hand, $\boldsymbol{x}_k$ happened to be a full variable, we relegate its treatment to the next part of the algorithm.

At the moment, we are considering a fixed term assignment, a fixed candidate assignment for the components, and several components being fixed to a value according to restrictions discussed in the previous paragraphs. Keeping all of said fixed values for each quantified variable, it remains only to be seen whether there is a way to assign the components that we have not fixed so far to values in such a way that all restrictions of the form $\textsc{Positive}(\boldsymbol{x}_i)$ hold. We can thus see the problem as, given a list of partial instances $\boldsymbol{x}'_i$ (according to the values fixed so far), decide whether there is positive completion $\boldsymbol{x}_i$ for each of them. This is actually the only step which we can do for FBDDs or perceptrons but that cannot be done efficiently for MLPs (unless PTIME = NP). By using Lemma 4 a constant number of times, we can decide if there is a completion $\boldsymbol{x}_i$ of $\boldsymbol{x}'$ that holds $\textsc{Positive}(\boldsymbol{x}_i)$ for every quantified variable $\boldsymbol{x}_i$, which was the only remaining step to verify whether the candidate assignment, and the fixed values, could satisfy all restrictions. If this is not the case, however, we can discard the assignment

We illustrate once again this last step on our running example.

Example 4 (Continuation of Example 3)**.** *We have* $\boldsymbol{x}'_1 = (0 \quad 1 \quad \bot \quad 1 \quad \bot)$*, and we need to decide whether there is a positive completion of* $\boldsymbol{x}'_1$*. As the model we are considering accepts instances that have at least 3 features with value* 1*, it is enough to consider the completion* $\boldsymbol{x}_1 = (0 \quad 1 \quad 1 \quad 1 \quad 0)$*. The reader can now check that* $\boldsymbol{x}_1$ *indeed satisfies the original formula.*

As each step of the described procedure can be done in polynomial time, and correctness is proved throughout the algorithm, we conclude the proof. □