Symbolic Knowledge Extraction and Injection with Sub-symbolic Predictors: A Systematic Literature Review

GIOVANNI CIATTO, Dipartimento di Informatica – Scienza e Ingegneria, ALMA MATER STUDIORUM—Università di Bologna, Cesena, Italy
FEDERICO SABBATINI, Dipartimento di Scienze Pure e Applicate, Università degli Studi di Urbino Carlo Bo, Urbino, Italy
ANDREA AGIOLLO, Dipartimento di Informatica – Scienza e Ingegneria, ALMA MATER STUDIORUM—Università di Bologna, Cesena, Italy
MATTEO MAGNINI, Dipartimento di Informatica – Scienza e Ingegneria, ALMA MATER STUDIORUM—Università di Bologna, Cesena, Italy
ANDREA OMICINI, Dipartimento di Informatica – Scienza e Ingegneria, ALMA MATER STUDIORUM—Università di Bologna, Cesena, Italy

In this article, we focus on the opacity issue of sub-symbolic machine learning predictors by promoting two complementary activities—*symbolic knowledge extraction* (SKE) and *symbolic knowledge injection* (SKI)—from and into sub-symbolic predictors. We consider as symbolic any language being intelligible and interpretable for both humans and computers. Accordingly, we propose general meta-models for both SKE and SKI, along with two taxonomies for the classification of SKE and SKI methods. By adopting an explainable artificial intelligence (XAI) perspective, we highlight how such methods can be exploited to mitigate the aforementioned opacity issue. Our taxonomies are attained by surveying and classifying existing methods from the literature, following a systematic approach, and by generalising the results of previous surveys targeting specific sub-topics of either SKE or SKI alone. More precisely, we analyse 132 methods for SKE and 117 methods for SKI, and we categorise them according to their purpose, operation, expected input/output data and predictor types. For each method, we also indicate the presence/lack of runnable software implementations. Our work may be of interest for data scientists aiming at selecting the most adequate SKE/SKI method for their needs, and may also work as suggestions for researchers interested in filling the gaps of the current state-of-the-art as well as for developers willing to implement SKE/SKI-based technologies.

CCS Concepts: • **Theory of computation** → **Logic**; *Machine learning theory*; • **Computing methodologies** → **Hybrid symbolic-numeric methods**; *Knowledge representation and reasoning;*

This paper was partially supported by *(*i) the CHIST-ERA IV project "EXPECTATION" (CHISTERA-19-XAI-005), funded by the Italian MUR; *(*ii) the "FAIR–Future Artificial Intelligence Research" project (PNRR, M4C2, Investimento 1.3, Partenariato Esteso PE00000013), funded by the European Commission under the NextGenerationEU programme; *(*iii) the "ENGINES – ENGineering INtElligent Systems around intelligent agent technologies" project funded by the Italian MUR program "PRIN 2022" under grant number 20229ZXBZM.

Authors' addresses: G. Ciatto, A. Agiollo, M. Magnini, and A. Omicini, Dipartimento di Informatica – Scienza e Ingegneria, Alma Mater Studiorum–Università di Bologna, via dell'Università 50, Cesena, Italy, 47522; e-mails: giovanni.ciatto@unibo.it, andrea.agiollo@unibo.it, matteo.magnini@unibo.it, andrea.omicini@unibo.it; F. Sabbatini, Dipartimento di Scienze Pure e Applicate, Università degli Studi di Urbino Carlo Bo, Via Santa Chiara, 27, Urbino, Italy, 61029; e-mail: f.sabbatini1@campus.uniurb.it.

ACM 0360-0300/2024/03-ART161
https://doi.org/10.1145/3645103

Additional Key Words and Phrases: Machine learning, logic, symbolic knowledge extraction, symbolic knowledge injection

ACM Reference Format:
Giovanni Ciatto, Federico Sabbatini, Andrea Agiollo, Matteo Magnini, and Andrea Omicini. 2024. Symbolic Knowledge Extraction and Injection with Sub-symbolic Predictors: A Systematic Literature Review. *ACM Comput. Surv.* 56, 6, Article 161 (March 2024), 35 pages. https://doi.org/10.1145/3645103

1 INTRODUCTION

In the context of **artificial intelligence (AI)**, an increasing number of critical applications that rely on **machine learning (ML)** are being developed. This promotes a data-driven approach to the engineering of intelligent computational systems in which hard-to-code tasks are (semi-)automatically learned from data rather than manually programmed by human developers. Tasks that can be learned this way range from text [42] to speech [41] or image recognition [59], stepping through time series forecasting, clustering, and so on. Applications are manifold and make our life easier in many ways—e.g., via speech-to-text applications, e-mail spam and malware filtering, customer profiling, automatic translation, virtual personal assistants, and so forth.

Learning, in particular, is automated via ML algorithms, often implying *numeric* processing of data—which, in turn, enables the detection of fuzzy patterns or statistically relevant regularities in the data that algorithms can learn to recognise. This is fundamental to supporting the automatic acquisition of otherwise hard-to-formalise behaviours for computational systems. However, flexibility comes at the cost of poorly *interpretable* solutions, as state-of-the-art *sub-symbolic* predictors—such as neural networks—are often exploited behind the scenes.

These predictors are commonly characterised by opacity [10, 32], as the interplay among the complexity of the data and the algorithms they are trained upon/with makes it hard for humans to understand their behaviour. Hence, by ‘interpretable’, we mean here that the expert human user may observe the computational system and understand its behaviour. Even though the property is not always required, there exist safety-, value-, or ethic-critical applications for which humans must be in full control of the computational systems supporting their decisions or aiding their actions. In those cases, the lack of interpretability is a no-go.

State-of-the-art ML systems rely on a collection of well-established data mining predictors, such as neural networks, support vector machines, decision trees, random forests, or linear models. Despite the latter sorts of predictors being often considered as interpretable in the general case, as the complexity of the problem at hand increases (e.g., dimensionality of the available data), trained predictors become more complex. Hence, they are harder to contemplate and, therefore, less interpretable. Nevertheless, these mechanisms have penetrated the modern practices of data scientists because of their flexibility and expected effectiveness in terms of predictive performance. Unfortunately, a number of experts have empirically observed an inverse proportionality relation among interpretability and predictive performance [13, 44]. This is the reason why data-driven engineering efforts targeting critical application scenarios nowadays have to choose between predictive performance and interpretability as their priority: we call this the *interpretability/performance trade-off*.

In this article, we focus on the problem of working around the interpretability/performance trade-off. We do so by promoting two complementary activities: ***symbolic knowledge extraction*** **(SKE)** and ***symbolic knowledge injection*** **(SKI)** from and into sub-symbolic predictors. In both cases, ‘symbolic’ refers to the way knowledge is represented. In particular, we consider as symbolic any language that is intelligible and interpretable for both human beings and computers. This includes a number of logic formalisms and excludes the fixed-sized tensors of numbers commonly exploited in sub-symbolic ML.

Intuitively, SKE is the process of distilling the knowledge a sub-symbolic predictor has grasped from data into symbolic form. This can be exploited to provide explanations for otherwise poorly interpretable sub-symbolic predictors. More generally, SKE enables the *inspection* of the sub-symbolic predictors it is applied to, making it possible for the human designer to figure out how they behave. Conversely, SKI is the inverse process of letting a sub-symbolic predictor follow the symbolic knowledge possibly encoded by its human designers. It enables a higher degree of *control* over a sub-symbolic predictor and its behaviour by constraining it with human-like common-sense—suitably encoded into symbolic form.

Apart from insights, notions such as SKE and SKI have rarely been described in general terms in the scientific literature despite the multitude of methods falling under their umbrellas. Hence, the aim of this article is to provide general definitions and descriptions of these topics other than providing durable taxonomies for categorising present and future SKE/SKI methods. Arguably, these contributions should take into account the widest possible portion of scientific literature to avoid subjectivity. Accordingly, in this article we propose a **systematic literature review (SLR)** following the three-folded purpose of (i) collecting and categorising existing methods for SKE and SKI into clear taxonomies, (ii) providing a wide overview of the state-of-the-art and technology, and (iii) detecting open research challenges and opportunities. In particular, we analyse 132 methods for SKE and 117 methods for SKI, classifying them according to their purpose, operation, expected input/output data and predictor types. For each method, we also probe the existence/lack of software implementations.

To the best of our knowledge, our survey is the only *systematic* work focusing on *both* SKE and SKI algorithms. Furthermore, with regard to other surveys on these topics, our SLR collects the greatest number of methods. In doing so, we elicit a meta-model for SKE (resp., SKI) according to which existing and future extraction (resp., injection) methods can be categorised and described. Our taxonomies may be of interest to data scientists willing to select the most adequate SKE/SKI method for their needs and may also work as suggestions for researchers interested in filling the gaps of the current state-of-the-art or developers willing to implement SKE or SKI software technologies.

Accordingly, the remainder of this article is organised as follows. Section 2 recalls the state-of-the-art for machine learning, symbolic AI, and **explainable AI (XAI)**, aimed at providing readers with a fast-track access to most of the concepts and terms used in the article. Section 3 delves into the details of what we mean by SKE and SKI and explains how this SLR is conducted: in that section, we declare our research questions and describe our research methodology. Section 4 answers our research questions, summarising the results of the analysis of the surveyed literature. The same results are then discussed in Section 5, in which major challenges and opportunities are elicited. Section 6 concludes the article.

2 BACKGROUND

2.1 Machine Learning

A widely adopted definition of machine learning by the author of [39] states:

> *. . . a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.*

This definition is very loose, as it does not specify (i) what are the possible tasks, (ii) how performance is measured in practice, (iii) how/when experience should be provided to tasks, (iv) how exactly the program is supposed learn, and (v) under which form learnt information is represented. Accordingly, depending on the particular ways these aspects are tackled, a categorisation of the approaches and techniques enabling software agents to learn may be drawn.

Three major approaches to ML exist: *supervised*, *unsupervised*, and *reinforcement* learning. Each approach is tailored on a well-defined pool of tasks that may, in turn, be applied in a wide range of use case scenarios. Accordingly, differences among these three approaches can be understood by looking at the sorts of tasks T they support, commonly consisting of the estimation of some unknown relation, and how experience E is provided to the learning algorithm.

In supervised learning, the learning task consists of finding a way to approximate an unknown relation given a sampling of its items that constitute the experience. In unsupervised learning, the learning task consists of finding the best relation for a sample of items that constitute the experience following a given optimality criterion intensionally describing the target relation. In reinforcement learning, the learning task consists of letting an agent estimate optimal plans given the reward it receives whenever it reaches particular goals. Here, the rewards constitutes the experience, whereas plans can be described as relations among the possible states of the world, the actions to be performed in those states, and the rewards the agents expect to receive from those actions.

Several practical AI problems—such as image recognition and financial and medical decision support systems—can be reduced to *supervised* ML—which can be further grouped in terms of either *classification* or *regression* problems [29, 48]. Within the scope of sub-symbolic supervised ML, a *learning algorithm* is commonly exploited to approximate the specific nature and shape of an unknown *prediction* function (or *predictor*) $\pi^* : \mathcal{X} \to \mathcal{Y}$, mapping data from an input space $\mathcal{X}$ into an output space $\mathcal{Y}$. Here, common choices for both $\mathcal{X}$ and $\mathcal{Y}$ are, for instance, the set of vectors, matrices, or tensors of numbers of a given size—hence, the sub-symbolic nature of the approach.

Without loss of generality, in the following we refer to items in $\mathcal{X}$ as n-dimensional vectors denoted as $\mathbf{x}$, whereas items in $\mathcal{Y}$ are m-dimensional vectors denoted as $\mathbf{y}$—matrices or tensors may be suitable choices as well.

To approximate function π^*, supervised learning assumes that a *learning algorithm* is in place. This algorithm computes the approximation by taking into account a number N of *examples* of the form $(\mathbf{x}_i, \mathbf{y}_i)$ such that $\mathbf{x}_i \in X \subset \mathcal{X}$, $\mathbf{y}_i \in Y \subset \mathcal{Y}$, and $|X| \equiv |Y| \equiv N$. Here, the set $D = \{(\mathbf{x}_i, \mathbf{y}_i) \mid \mathbf{x}_i \in X, \mathbf{y}_i \in Y\}$ is called a *training* set, which consists of $(n + m)$-dimensional vectors. The dataset can be considered as the concatenation of two matrices, namely the $N \times n$ matrix of *input* data (X) and the $N \times m$ matrix of *expected output* data (Y). Here, each $\mathbf{x}_i$ represents an instance of the input data for which the expected output value $\mathbf{y}_i \equiv \pi^*(\mathbf{x}_i)$ is known or has already been estimated. Notably, these types of ML problems are said to be 'supervised' *because* the expected outputs Y are available. The function approximation task is called **regression** if the components of Y consist of continuous or numerable—i.e., *infinite*—values, and called **classification** if they consist of categorical—i.e., *finite*—values.

2.1.1 On the Nature of Sub-symbolic Data. ML methods, and sub-symbolic approaches in general, represent data as (possibly multi-dimensional) *arrays* (e.g., vectors, matrices, or tensors) of real numbers and knowledge as functions over data. This is particularly relevant as opposed to *symbolic* knowledge representation approaches, which represent data via logic formulæ (see Section 2.2).

In spite of the fact that numbers are technically symbols as well, we cannot consider arrays and their functions as means for symbolic **knowledge representation (KR)**. Indeed, according to [50], to be considered as symbolic, KR approaches should (a) involve a set of symbols (b) that can be combined (e.g., concatenated) in possibly infinite ways following precise grammatical rules and

(c) where both elementary symbols and any admissible combination of them can be assigned with *meaning*—i.e., each symbol can be mapped into some entity from the domain at hand. Below, we discuss how sub-symbolic approaches most typically do not satisfy requirements 2.1.1 and 2.1.1.

Vectors, matrices, tensors. Multi-dimensional arrays are the basic brick of sub-symbolic data representation. More formally, a D-order array consists of an ordered container of real numbers, where D denotes the amount of indices required to locate each single item into the array. We may refer to 1-order arrays as *vectors*, 2-order arrays as *matrices*, and higher-order arrays as *tensors*.

In any given sub-symbolic data-representation task leveraging upon arrays, information may be carried by both (i) the actual numbers contained into the array, and (ii) their location into the array itself. In practice, the actual dimensions ($d_1 \times \ldots \times d_D$) of the array play a central role as well. Indeed, sub-symbolic data processing is commonly tailored on arrays of *fixed* sizes—meaning that the actual values of $d_1, \ldots, d_D$ are chosen at design time and never changed after that. This violates requirement 2.1.1 above, hence, we define sub-symbolic KR as the task of expressing data in the form of *rigid* arrays of *numbers*.

Local vs. distributed. When data is represented in the form of numeric arrays, the whole representation may be *local* or *distributed* [50]. In local representations, each single number into the array is characterised by a well-delimited meaning—i.e., it is measuring or describing a clearly identifiable concept from a given domain. Conversely, in distributed representations, each single item of the array is nearly meaningless, unless it is considered along with its neighbourhood—i.e., any other item that is 'close' in the indexing space of the array according to some given notion of closeness. Thus, while in local representations the location of each number in the array is mostly negligible, in distributed representations it is of paramount importance. Notably, distributed representations violate the aforementioned requirement 2.1.1. In recent literature, authors call 'sub-symbolic' those predictors who rely on distributed representations of data.

2.1.2 *Overview on ML Predictors.*

Depending on the predictor family of choice, the nature of the admissible hypothesis spaces and learning algorithms may vary dramatically as well as the predictive performance of the target predictor, and the whole efficiency of learning.

In the literature of machine learning, statistical learning, and data mining, a plethora of learning algorithms have been proposed through the years. Because of the **'no free lunch' (NFL)** theorem [55], however, no algorithm is guaranteed to outperform the others in all possible scenarios. For this reason, the literature and the practice of data science keeps leveraging on algorithms and methods whose first proposal was published decades ago. The most notable algorithms include, among the many others, (deep) **neural networks (NNs)**, **decision trees (DTs)**, (generalised) linear models, nearest neighbours, **support vector machines (SVMs)**, and random forests.

These algorithms can be categorised in several ways, for instance, depending (i) on the supervised learning task they support (classification vs. regression) or (ii) on the underlying strategy adopted for learning (e.g., gradient descent, least square optimisation).

Some learning algorithms (e.g., NNs) naturally target regression problems despite being adaptable to classification as well, whereas others (e.g., SVMs) target classification problems while being adaptable to regression as well. Similarly, some target multi-dimensional outputs ($\mathbf{y} \in \mathbb{R}^m$ and $m > 1$), whereas others target mono-dimensional outputs ($m = 1$). Regressors are considered as the most general case, as other learning tasks can usually be defined in terms of mono-dimensional regression.

The learning strategy is inherently bound to the predictor family of choice. NNs, for instance, are trained via back-propagation [46] and **stochastic gradient descent (SGD)**, generalised linear models via Gauss's least squares method, decision trees via methods described in [9], and so forth.

Even though all the aforementioned algorithms may appear interchangeable in principle because of the NFL theorem, their malleability is very different in practice. For instance, the least square method involves inverting matrices of order N, where N is the amount of available examples in the training set, making the computational complexity of learning more than quadratic in time. Furthermore, in practice, convergence of the method is not guaranteed in the general case; instead, it is guaranteed for generalised linear models, hence it is not adopted elsewhere. Thus, learning by least square optimisation may become impractical for big datasets or for predictor families outside the scope of generalised linear models. Conversely, the SGD method involves arbitrarily sized subsets of the dataset (i.e., batches) to be processed a finite (i.e., controllable) amount of times. Hence, the complexity of SGD can be finely controlled and adapted to the computational resources at hand, e.g., by making the learning process incremental and by avoiding all data to be loaded in memory. Moreover, SGD can be applied to several sorts of predictor families (including NNs and generalised linear models), as it only requires the target function to be differentiable with regard to its parameters. For all these reasons, despite the lack of optimality guarantees, SGD is considered to be very effective, scalable, and malleable in practice. Hence, it is extensively exploited in modern data science applications.

In the remainder of this subsection, we focus on two families of predictors, DTs and NNs, and their respective learning methods. We focus precisely on them because they are related to many surveyed SKE/SKI methods. DTs are noteworthy because of their user friendliness, whereas NNs are mostly popular because of their predictive performance and flexibility.

Decision trees. Decision trees are particular sorts of predictors supporting both classification and regression tasks. In their learning phase, the input space is recursively *partitioned* through a number of splits (i.e., *decisions*) based on the input data X in such a way that the prediction in each partition is constant and the error with regard to the expected outputs Y is minimal while keeping the total amount of partitions low as well. The whole procedure then synthesises a number of *hierarchical* decision rules to be followed whenever the prediction corresponding to any $x \in \mathcal{X}$ must be computed. In the inference phase, decision rules are orderly evaluated from the root to a leaf, to select the portion of the input space $\mathcal{X}$ containing x. As each leaf corresponds to a single portion of the input space, the whole procedure results in a single prediction for each x.

Unlike other families of predictors, the peculiarity of DTs lies in the particular outcome of the learning process—that is, the *tree* of decision rules—which is straightforwardly intelligible for humans and graphically representable in 2D charts. As further discussed in the remainder of the article, this property is of paramount importance whenever the inner operation of an automatic predictor must be interpreted and understood by a human agent.

Neural networks. Neural networks are biologically inspired computational models made of several elementary units (neurons) commonly interconnected into a **directed acyclic graph (DAG)** via *weighted* synapses. Accordingly, the most relevant aspects of NNs concern the inner operation of neurons and the particular architecture of their interconnection.

Neurons are very simple numeric computational units. They accept n scalar inputs $(x_1, \ldots, x_n) = \mathbf{x} \in \mathbb{R}^n$ weighted by as many scalar weights $(w_1, \ldots, w_n) = \mathbf{w} \in \mathbb{R}^n$ and they process the linear combination $\mathbf{x} \cdot \mathbf{w}$ via an activation function $\sigma : \mathbb{R} \mapsto \mathbb{R}$, producing a scalar output $y = \sigma(\mathbf{x} \cdot \mathbf{w})$. The output of a neuron may become the input of many others, possibly forming *networks* of neurons having arbitrary topologies. These networks may be fed with any numeric information encoded as vectors of real numbers by simply letting a number of neurons produce constant outputs.

While virtually all topologies are admissible for NNs, not all are convenient. Many convenient *architectures*—roughly, patterns of well-studied topologies—have been proposed in the literature [51] to serve disparate purposes far beyond the scope of supervised machine learning. However, identification of the most appropriate architecture for any given task is non-trivial: recent efforts propose to learn their construction automatically [2, 33].

Most common NN architectures are feed-forward, meaning that neurons are organised in *layers*, where neurons from layer i can only accept ingoing synapses from neurons of layers $j < i$. The first layer is considered the input layer, which is used to *feed* the whole network. The last one is the output layer, where predictions are drawn. In NN architectures, inference lets information flow from the input to the output layer assuming the weights of synapses are fixed, whereas training lets information flow from the output to the input layer, causing the variation of weights to minimise the prediction error of the overall network.

The recent success of deep learning [20] has proved the flexibility and the predictive performance of ***deep* neural networks (DNNs)**. 'Deep' here refers to the large amount of (possibly *convolutional*) layers. In other words, DNNs can learn how to apply cascades of convolutional operations to the input data. Convolutions let the network spot relevant features in the input data, at possibly different scales. This is why DNNs are good at solving complex pattern-recognition tasks—e.g., computer vision or speech recognition. However, unprecedented predictive performances of DNNs come at the cost of their increased internal complexity, non-inspectability, and greater data greediness.

2.1.3 General Supervised Learning Workflow. Briefly speaking, an ML workflow is the process of producing a suitable predictor for the available data and the learning task at hand with the purpose of exploiting the predictor later to draw analyses or to drive decisions. Hence, any ML workflow is commonly described as composed of two major phases: training, in which predictors are fitted on data, and inference, in which predictors are exploited. However, in practice, further phases are included, such as data provisioning and pre-processing as well as model selection and assessment.

In other words, before using a sub-symbolic predictor in a real-world scenario, data scientists must ensure that it has been sufficiently trained and its predictive performance is sufficiently high. In turn, training requires (i) an adequate amount of data to be available; (ii) a family of predictors to be chosen (e.g., NNs, K-nearest neighbours, linear models); (iii) any structural hyper-parameter to be defined (e.g., amount, type, size of layers, K, maximum order of the polynomials); (iv) and any other learning parameter to be fixed (e.g., learning rate, momentum, batch size, epoch limit). Data must therefore be provisioned before training and possibly pre-processed to ease training itself, for example, by normalising data or by encoding non-numeric features into numeric form. The structure of the network must be defined in terms of (roughly) input, hidden, and output layers as well as their activation functions. Finally, hyper-parameters must be carefully tuned according to the data scientist's experience and the time constraints and computational resources at hand.

Thus, from a coarse-grained perspective, an ML workflow can be conceived as composed of six major phases, enumerated as follows.

(1) **Sub-symbolic data gathering:** The first actual step of any ML workflow, in which data is loaded in memory for later processing
(2) **Pre-processing:** The application of several bulk operations to the training data, following several purposes, such as (i) homogenise the variation ranges of the many features sampled by the dataset, (ii) detect irrelevant features and remove them, (iii) construct relevant features by combining the existing ones, or (iv) encoding non-numeric features into numeric form
(3) **Predictor selection:** A principled search for the most adequate sort of predictor to tackle the data and the learning task at hand, which is where hyper-parameters are commonly fixed
(4) **Training:** The actual tuning of the selected predictor(s) on the available data, which is where parameters are commonly fixed

(5) **Validation:** Measuring the predictive performance of trained predictors, with the purpose of assessing whether and to what extent it will generalise to new, unseen data
(6) **Inference:** The final phase, in which trained predictors are used to draw predictions on unknown data—that is, different data with regard to the one used for training

2.2 Computational Logic

Symbolic KR has always been regarded as a key issue since the early days of AI, as no intelligence can exist without knowledge and no computation can occur in lack of representation. When compared with arrays of numbers, symbolic KR is far more flexible, expressive, and, in particular, more intelligible. It is both machine- and human-interpretable. Historically, most KR formalisms and technologies have been designed on top of *computational logic* [34], that is, the exploitation of formal logic in computer science. Consider, for instance, *deductive databases* [23], *description logics* [5], *ontologies* [17], *Horn* logic [37], and *higher-order* logic [49], just to name a few.

2.2.1 Formal Logics. Many kinds of logic-based KR systems have been proposed over the years, mostly relying on ***first-order logic*** **(FOL)** either by restricting or extending it, e.g., on description logics and modal logics, which have been used to represent, for instance, terminological knowledge and time-dependent or subjective knowledge. Here, we briefly recall the state-of-the-art of FOL and its most relevant subsets.

First-order logic. FOL is a general-purpose logic that can be used to represent knowledge symbolically, in a very flexible way. More precisely, it allows both human and computational agents to express (i.e., write) the properties of, and the relations among, a set of entities constituting the *domain of the discourse* via one or more *formulæ* and, possibly, to reason over such formulæ by drawing inferences. Here, the domain of the discourse $\mathbb{D}$ is the set of all relevant entities that should be represented in FOL to be amenable of formal treatment in a particular scenario.

Informally, the syntax for the general FOL formula is defined over the assumption that there exist: (i) a set of *constant* or *function* symbols, (ii) a set of *predicate symbols*, and (iii) a set of *variables*. Under this assumption, a FOL formula is any expression composed of a list of quantified variables, followed by a number of *literals*, i.e., *predicates* that may or may not be prefixed by the negation operator (¬). Literals are commonly combined into expressions via *logic connectives*, such as conjunction (∧), disjunction (∨), implication (→), or equivalence (↔).

Each predicate consists of a predicate symbol, possibly applied to one or more *terms*. Terms may be of three sorts: *constants*, *functions*, or *variables*. Constants represent entities from the domain of the discourse. In particular, each constant references a different entity. Functions are combinations of one or more entities via a *function symbol*. Similar to predicates, functions may carry one or more terms. Being containers of terms, functions enable the creation of arbitrarily complex data structures combining several elementary terms into composite ones. This kind of composability by recursion is what makes the aforementioned definition of 'symbolic' valid for FOL. Finally, variables are placeholders for unknown terms, i.e., for either individual entities or groups of entities.

Predicates and terms are very flexible tools to represent knowledge. While terms can be used to represent or reference either entities or groups of entities from the domain of the discourse, predicates can be used to represent relations among entities or the properties of each single entity.

Intensional vs. extensional. In logic, one may define concepts—i.e., describe data—either *extensionally* or *intensionally*. Extensional definitions are *direct* representations of data. In the particular case of FOL, this implies defining a relation or set by explicitly mentioning the entities it involves. Conversely, intensional definitions are *indirect* representations of data. In the particular case of FOL, this implies defining a relation or set by describing its elements via other relations or sets.

Recursive intensional predicates are very expressive and powerful, as they enable the description of infinite sets via a finite (and commonly small) amount of formulæ. This is one of the key benefits of FOL as a means for KR.

2.2.2 Expressiveness vs. Tractability: Notable Subsets of FOL. Tractability deals with the theoretical question: Can a logic reasoner compute whether a logic formula is true (or not) in *reasonable* time? Such aspects are deeply entangled with the particular reasoner of choice. Depending on which and how many features a logic includes, it may be more or less *expressive*. The higher the expressiveness, the more the complexity of the problems that may be represented via logic and processed via inference increases. This opens the possibility for the solver to meet queries that cannot be answered in practical time or by relying upon a limited amount of memory—or just cannot get an answer at all. Roughly speaking, more expressive logic languages make it easier for human beings to describe a particular domain, usually requiring them to write less and more concise clauses at the expense of higher difficulty for software agents to draw inferences autonomously, because of computational tractability. This is a well-understood phenomenon in both computer science and computational logic [8, 31], often referred to as the *expressiveness/tractability* trade-off.

FOL, in particular, is considered very expressive. Indeed, it comes with many undecidable, semi-decidable, or simply intractable properties. Hence, several relevant subsets of FOL have been identified in the literature, often sacrificing expressiveness for tractability. Major notions concerning these logics are recalled below.

Horn logic. Horn logic is a notable subset of FOL, characterised by a good trade-off among theoretical expressiveness and practical tractability [36].

Horn logic is designed around the notion of the *Horn clause* [26]. Horn clauses are FOL formulæ having no quantifiers and consisting of a disjunction of predicates, where only at most one literal is non-negated—or, equivalently, an implication having a single predicate as post-condition and a conjunction of predicates as pre-condition: $h \leftarrow b_1, \ \ldots, \ b_n$. Here, ← denotes logic implication from right to left, commas denote logic conjunction, and all b_i, as well as h, are predicates of arbitrary arity, possibly carrying FOL terms of any sort—i.e., variables, constants, or functions. Horn clauses are thus if–then rules written in reverse order and only supporting conjunctions of predicates as pre-conditions.

Essentially, Horn logic is a very restricted subset of FOL where (i) formulæ are reduced to clauses, as they can only contain predicates, conjunctions, and a single implication operator; therefore, (ii) operators such as $\vee$, $\leftrightarrow$, or $\neg$ cannot be used; (iii) variables are implicitly quantified; and (iv) terms work as in FOL.

Datalog. Datalog is a restricted subset of FOL [3] representing knowledge via function-free Horn clauses, defined above. Thus, essentially, Datalog is a subset of Horn logic where structured terms (i.e., recursive data structures) are forbidden. This is a direct consequence of the lack of function symbols.

Similar to Horn logic, Datalog's knowledge bases consist of sets of function-free Horn clauses.

Description logics. **Description logics (DL)** are a family of subsets of FOL, generally involving some or no quantifiers, no structured terms, and no n-ary predicates such that $n \geq 3$. In other words, description logics represent knowledge by only leveraging on constants and variables other than atomic, unary, and binary predicates.

Differences among specific variants of DL lay in which and how many logic connectives are supported other than, of course, whether negation is supported or not. The wide variety of DL is due to the well-known expressiveness/tractability trade-off. However, depending on the particular situation at hand, one may either prefer a more expressive (≈feature-rich) DL variant at the price of a reduced tractability (or even decidability) of the algorithms aimed at manipulating knowledge represented through that DL or *vice versa.*

Regardless of the particular DL variant of choice, it is common practice in the scope of DL to call (i) constant terms 'individuals' as each constant references a single entity from a given domain, (ii) unary predicates, e.g., either 'classes' or 'concepts' as each predicate *groups* a set of individuals, i.e., all those individuals for which the predicate is true, and (iii) binary predicates, e.g., either 'properties' or 'roles' as each predicate *relates* two sets of individuals. Following such a nomenclature, any piece of knowledge can be represented in DL by tagging each relevant entity with some constant (e.g., a URL) and by defining concepts and properties accordingly.

Notably, binary predicates are of particular interest as they support connecting couples of entities altogether. This is commonly achieved via subject–predicate–object *triplets*, i.e., ground binary predicates of the form $\langle \mathsf{a}\ f\ \mathsf{b} \rangle$ or $f(\mathsf{a}, \mathsf{b})$, where a is the subject, f is the predicate, and b is the object. Such triplets allow users to *extensionally* describe knowledge in a readable, machine-interpretable, and tractable way.

Collections of triplets constitute the so-called ***knowledge graphs*** **(KGs)**, i.e., directed graphs where vertices represent individuals, while arcs represent the binary properties connecting these individuals. These may explicitly or implicitly instantiate a particular *ontology*, i.e., a formal description of classes characterising a given domain and description of their relations (inclusion, exclusion, intersection, equivalence, etc.) as well as the properties they must (or must not) include.

Propositional logic. Propositional logic is a very restricted subset of FOL, where quantifiers, terms, and non-atomic predicates are missing. Hence, propositional formulæ simply consist of expressions involving one or many 0-ary predicates—i.e., *propositions*—possibly interconnected by ordinary logic connectives. Here, each proposition may be interpreted as a Boolean variable that can either be true or false and the truth of formulæ can be computed as in the Boolean algebra. Thus, for instance, a notable example of a propositional formula could be as follows: $p \land \neg q \rightarrow r$, where p may be the proposition 'it is raining', q may be the proposition 'there is a roof', and r may be the proposition 'the floor is wet'.

The expressiveness of propositional logic is far lower than the one of FOL. For instance, because of the lack of quantifiers, each relevant aspect/event should be explicitly modelled as a proposition. Furthermore, because of the lack of terms, entities from a given domain cannot be explicitly referenced. Such a lack of expressiveness, however, implies that computing the *satisfiability* of a propositional formula is a *decidable* problem, which may be a desirable property in some application scenarios.

Despite the fact that propositional logic may appear too trivial to handle common decision tasks where non-binary data is involved, it turns out that a number of apparently complex situations can indeed be reduced to a propositional setting. This is the case, for instance, of any expression involving numeric variables or constants, arithmetical comparison operators, logic connectives, and nothing more than that. In fact, formulæ containing comparisons among variables or constants (or among each others) can be reduced to propositional logic by mapping each comparison into a proposition.

2.3 eXplainable Artificial Intelligence

Modern intelligent systems are increasingly adopting *sub-symbolic* predictive models to support their intelligent behaviour. These are commonly trained following a data-driven approach. Such wide adoption is unsurprising given the unprecedented availability of data characterising the last decade. ML algorithms enable the detection of useful statistical information buried in data semi-automatically. Information, in turn, supports decision-making, monitoring, planning, and forecasting virtually in any human activity where data is available.

However, despite its predictive capabilities, ML comes with some drawbacks making it perform poorly in critical use cases. The most relevant example is algorithmic *opacity*—intuitively, the human struggle to *understand* how ML-based systems operate or take their decisions. In particular, we refer to 'opacity' according to the third definition provided by Burrell [10]: "opacity as the way algorithms operate at the scale of application". In ML-based applications, complexity—and, therefore, opacity—arises because of the hardly predictable interplay among highly dimensional datasets, the algorithms processing them, and the way such algorithms may change their behaviour during learning.

Opacity is a serious issue in all those contexts in which human beings are liable for their decisions or when they are expected/required to provide some sort of *explanation* for them—even if a decision has been suggested by software systems. This may be the case, for instance, in the healthcare, financial, or legal domains. In such contexts, ML is at the same time both an enabling factor, as it automates decision-making, and a limiting one, as opacity reduces human *control* on decision-making. The overall effect is general *distrust* with regard to AI-based solutions.

Opacity is also the reason why ML predictors are called 'black boxes' in the literature. The expression refers to systems in whic knowledge is not symbolically represented [32]. In absence of symbolic representations, *understanding* the operation of black boxes, or why they recommend or take particular decisions, becomes hard for humans. The inability to understand black-box content and operation may then prevent people from fully trusting (and, therefore, accepting) them.

To make the picture even more complex, current regulations such as the **General Data Protection Regulation (GDPR)** [52] are starting to recognise the citizens' *right to explanation* [21]—which eventually mandates *understandability* of intelligent systems. This step is essential to guarantee algorithmic fairness, to identify potential biases/problems in the training data or in the black box's operation, and to ensure that intelligent systems work as expected.

Unfortunately, to date, the notion of understandability is neither standardised nor systematically assessed. No consensus has been reached on what 'providing an explanation' should mean when decisions are supported by ML [38]. However, many authors agree that black boxes are not equally *opaque*: some are more susceptible to interpretation than others for our minds. For example, Figure 1 shows how differences in black-box interpretability are conventionally described.

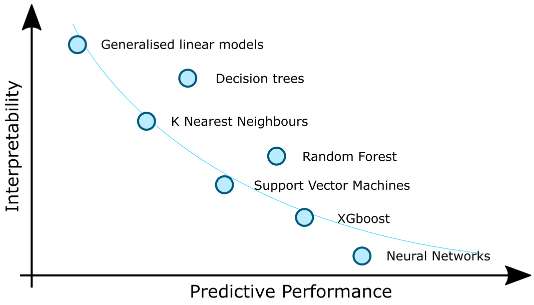


Fig. 1. Interpretability/performance trade-off for some common sorts of black-box predictors.

Despite being informal, as argued by [44], given the lack of measures for 'interpretability', Figure 1 effectively expresses why research on understandability is needed. The figure stresses how the better-performing black boxes are also the less interpretable ones. This is troublesome as, in practice, predictive performance can only rarely be preferred over interpretability.

Nevertheless, consensus has been reached about *interpretability* and *explainability* being desirable properties for intelligent systems. Hence, within the scope of this article, we may briefly and informally describe XAI as the corpus of literature and methods aimed at making sub-symbolic AI more interpretable for humans, possibly by automating the production of explanations.

Along this line, based on the preliminary work featured in [15, 16] and by drawing inspiration from computational logic (in particular, model theory), we let ‘interpretation’ indicate “*the subjective relation that associates each representation with a specific meaning in the domain of the problem*”. In other words, interpretability refers to the cognitive effort required by human observers to assign a meaning to the way intelligent systems work or motivate the outcomes they produce. In those contexts, the notion of interpretability is often coupled with properties as algorithmic transparency (characterising approaches that are *not* opaque), decomposability, or simulatability-in a nutshell, *predictability.* Essentially, interpretable systems are understandable when humans can predict their behaviour.

As far as the term *explanation* is concerned, we trace back its meaning to Aristotelian thought beyond the Oxford dictionary definition, which defines *explanation* as “*a set of statements or accounts that make something clear, or, alternatively, the reasons or justifications given for an action or belief.*” Thus, an explanation is an *activity* aimed at making the relevant details of an object clear or easy to understand to an observer.

Accordingly, the concepts of explainability and interpretability are basically *orthogonal.* However, they are not unrelated: explanations may consist of constructing better (≈more interpretable) representations for the black box at hand.

This is the case, for instance, of “explanation by model simplification” [47], in which a poorly interpretable model is translated into a more interpretable one, having “high fidelity” [24] with the first one. The translation process of the first model into the second one can be considered as an explanation. For example, as surveyed by this article, several methods exist for *extracting* symbolic knowledge out of sub-symbolic predictors. When this is the case, the extraction act is technically an explanation, as it produces (more) interpretable objects—the symbolic knowledge—out of (less) interpretable ones—the predictors.

Conversely, one may regulate the interpretability of an opaque model by altering it to become ‘consistent’ with (i.e., ‘behave like’) some more interpretable one. In this case, no explanation is involved, yet the resulting model has a higher degree of interpretability—which is commonly the goal. For instance, as discussed in this article, several methods exist for *injecting* symbolic knowledge into sub-symbolic predictors. When this is the case, the injection acts as the means by which opacity issues are worked around.

Interpretability and explainability are key enabling properties for making AI-based solutions (more) trustworthy in the eyes of human users. However, as highlighted by Rudin et al. [45], they are not necessarily sufficient: they may also enable *distrust.* In other words, interpretability and explainability enable finer control on intelligent systems, letting users decide whether to trust them or not. Along this line, the surveyed SKE/SKI methods should be regarded as tools for increasing the degree of control that users have on AI systems.

2.3.1 Sorts of Explanation. According to the main impact surveys in the XAI area [6, 12, 24], two major approaches exist to bring explainability or interpretability features to intelligent systems: *by design* or *post-hoc.*

XAI by design. This approach to XAI aims at making intelligent systems interpretable or explainable *ex-ante* since they are designed to keep these features as first-class goals. Methods adhering to this approach can be further classified according to two sub-categories.

- **Symbols as constraint:** Containing methods supporting the creation of predictive models, possibly including or involving some black-box components, whose behaviour is constrained by a number of symbolic and intelligible rules, usually expressed in terms of (some subset of) FOL.

Transparent box design: Containing methods supporting the creation of predictive models that are inherently interpretable, requiring no further manipulation.

In the remainder of this article, we focus on methods from the latter category as it is deeply entangled with symbolic knowledge *injection*.

Post-hoc explainability. This approach to XAI aims at making intelligent systems interpretable or explainable *ex-post*, i.e., by somehow manipulating poorly interpretable pre-existing systems. Methods adhering to this approach can be further classified according to the following subcategories.

Text explanation: In which explainability is achieved by generating textual explanations that help to explain the model results; methods that generate symbols representing the model behaviour are also included in this category, as symbols represent the logic of the algorithm through appropriate semantic mapping.

Visual explanation: Techniques that allow the visualisation of the model behaviour; several techniques existing in the literature come along with methods for dimensionality reduction to make visualisation human interpretable.

Local explanation: In which explainability is achieved by first segmenting the solution space into less complex solution subspaces relevant for the whole model, then producing their explanation.

Explanation by example: Allows for the extraction of representative examples that capture the internal relationships and correlations found by the model.

Model simplification: Techniques allowing the construction of a completely new simplified system, trying to optimise similarity with the previous one while reducing complexity.

Feature relevance: Methods focus on how a model works internally by assigning a relevance score to each of its features, thus revealing their importance for the model in the output.

In the remainder of this article, we focus on methods from the 'model simplification' category, as it is deeply entangled with symbolic knowledge *extraction*.

3 DEFINITIONS AND METHODOLOGY

The goal of our SLR is to detect and categorise the many SKE and SKI algorithms proposed in the literature so far, hence shaping a clear picture of what SKE and SKI mean today.

Following this purpose, we start from broad and intuitive definitions of both SKE and SKI (provided in Section 3.1); we then (i) define a number of research questions aimed at delving into the details of actual SKE and SKI methods; along this line, we (ii) explore the literature looking for contributions matching the broad definitions from step (i) (following a strategy described in Section 3.2). Finally, by analysing such contributions, we (iii) provide answers for the research questions from step 3 (in Section 4) and, in doing so, we (iv) synthesise general, bottom-up taxonomies for both SKE and SKI (in Sections 4.1 and 4.2).

3.1 Definitions for Symbolic Knowledge Extraction and Injection

Here, we provide broad definitions for both symbolic knowledge extraction and injection, following the purpose of drawing a line among what methods, algorithms, and technologies from the literature should be considered related to either SKE or SKI and what should not. We do so with an XAI perspective, highlighting how both SKE and SKI help mitigate the opacity issues arising in data-driven AI. Then, we discuss the potential of the *joint* exploitation of SKE and SKI.

We fine-tune our definitions to comprehend and generalise the many methods and algorithms surveyed later in this article. Looking for a wider degree of generality, our definitions commit to no particular form of symbolic knowledge or sub-symbolic predictor despite the fact that many surveyed techniques come with commitments of that sort. Hence, in what follows we use ‘symbolic knowledge’ to mean ‘any chunk of intelligible information expressed in *any* possibly sort of logic’ as well as any sort of information that can be rewritten in logic form (e.g., decision trees). Similarly, we use ‘sub-symbolic predictor’ to mean ‘any sort of *supervised* ML model that can be fitted over *numeric* data to eagerly solve classification or regression tasks’.

3.1.1 Extraction. Generally speaking, SKE serves the purpose of generating intelligible representations for the sub-symbolic knowledge that an ML predictor has grasped from data during learning. Here, we provide a general definition of SKE and discuss its purpose as well as the major benefits it brings against the XAI landscape.

Definition. We define SKE as

> *any* algorithmic *procedure accepting* trained *sub-symbolic predictors as input and producing* symbolic *knowledge as output so that the extracted knowledge reflects the behaviour of the predictor with high* fidelity.

This definition emphasises a number of key aspects of SKE that are worth describing in further detail.

First, SKE is modelled as a class of *algorithms*—hence, finite-step recipes—characterised by what they accept as input and what they produce as output.

As far as the inputs of SKE procedures are concerned, the only explicit requirement is on *trained* ML predictors. There is no constraint with regard to the nature of the predictor itself. Hence, SKE procedures may be designed for any possible predictor family in principle. Yet, this requirement implies that the predictor’s training has already occurred and has reached some satisfying performance with regard to the task it has been trained for. Hence, in an ML workflow, SKE should occur *after* training and validation are concluded.

As far as the outputs of SKE procedures are concerned, the only explicit requirement is about the production of *symbolic* knowledge. ‘Symbolic’ is intended here, in a broader sense, as a synonym of ‘intelligible’ (for the human being). Hence, admissible outcomes are logic formulæ as well as decision trees or bare human-readable text.

In any case, for an algorithm to be considered a valid SKE procedure, the output knowledge should mirror as much as possiblethe behaviour of the original predictor with regard to the domain it was trained for. This involves a fidelity score aimed at measuring how well the extracted knowledge mimics the predictor it was extracted by with regard to the domain and the task that predictor was trained for. This, in turn, implies that the extracted knowledge should act in principle as a predictor as well, thus being as queryable as the original predictor would be. Thus, for instance, if the original predictor is an image classifier, the extracted knowledge should let an intelligent agent classify images of the same sort, expecting the same result. The agent may then be either computational (i.e., a software program) or human depending on whether the extracted knowledge is machine- or human-interpretable. The exploitation of *logic* knowledge as the target of SKE is of particular interest as it would enable both options.

Purpose and benefits. Generally speaking, one may be interested in performing SKE to inspect the inner operation of an opaque predictor, which should be considered a black box otherwise. However, one may also perform SKE to automatise and speed up the process of acquiring symbolic knowledge instead of crafting knowledge bases manually.

Inspecting a black-box predictor through SKE, in turn, is an interesting capability within the scope of XAI. Given a black-box predictor and a knowledge-extraction procedure applicable to it, any extracted knowledge can be adopted as a basis to construct explanations for that particular predictor. The extracted knowledge may act as an *interpretable replacement* (i.e., surrogate model) for the original predictor, provided that the two have a high-fidelity score [15].

Accordingly, the application of SKE to XAI brings a number of relevant opportunities, e.g., by letting human users (i) study the internal operation of an opaque predictor to find mispredicted input patterns or correctly predicted input patterns leveraging some unethical decision process; (ii) highlight the differences or the common behaviours between two or more black-box predictors performing the same task; and (iii) merge the knowledge acquired by various predictors, possibly of different kinds, on the same domain provided that the same representation format is used for extraction procedures [14].

3.1.2 Injection. Generally speaking, SKI serves a dual purpose with regard to SKE. SKI aims at letting an ML predictor take some symbolic knowledge into account when drawing predictions. Here, we provide a general definition of SKI and discuss its purpose and the major benefits it brings with regard to the XAI panorama.

Definition. We define SKI as

> *any* algorithmic *procedure affecting how sub-symbolic predictors draw their inferences in such a way that predictions are either* computed *as a function of, or made* consistent *with, some* given *symbolic knowledge.*

This definition emphasises a number of key aspects of SKI that are worth describing in further detail. Similar to SKE, it is modelled as a class of *algorithms.* Yet, dually with regard to extraction, SKI algorithms are procedures accepting symbolic knowledge as input and producing ML predictors as output.

In terms of the inputs of SKI procedures, the only explicit requirement is that knowledge should be symbolic and user provided—hence, *human* interpretable. However, since any input knowledge should be algorithmically manipulated by the SKI procedure, we elicit an implicit requirement here, constraining the input knowledge to be *machine* interpretable as well. This implies that some formal language—e.g., some formal logic or decision tree—should be employed for knowledge representation, whereas free text or natural language should be avoided.

Along this line, another implicit requirement is that the input knowledge should be *functionally analogous* with regard to the predictors undergoing injection. In other words, if a predictor aims at classifying customer profiles as either worthy or unworthy for credit, then the symbolic knowledge should encode decision procedures to serve the exact same purpose and observe the exact same information.

In terms of the outcomes of SKI procedures, our definition identifies two relevant situations that are not necessarily mutually exclusive. On the one hand, SKI procedures may enable sub-symbolic predictors to accept symbolic knowledge as input. SKI procedures of this sort essentially consist of a pre-processing algorithm aimed at encoding symbolic knowledge in sub-symbolic form, enabling sub-symbolic predictors to accept them as input. In this sense, SKI procedures of this sort enable sub-symbolic predictors to (learn how to) *compute* predictions as functions of the symbolic knowledge they were fed with assuming that it has been conveniently converted into sub-symbolic form. On the other hand, SKI procedures may alter sub-symbolic predictors so that they draw predictions that are *consistent* with the symbolic knowledge according to some notion of *consistency.* SKI procedures of this sort essentially affect either the structure or the training process of the sub-symbolic predictors they are applied to in such a way that the predictor must then take the symbolic knowledge into account when drawing predictions. In this sense, SKI procedures of this sort force sub-symbolic predictors to learn not only from data but from symbolic knowledge as well.

In any case, regardless of their outcomes, SKI procedures fit the ML workflow in its early phases, as they may affect both pre-processing and training.

Notably, consistency plays a pivotal role in SKI, dually with regard to what fidelity does for SKE. Along this line, our definition involves a consistency score aimed at measuring how well the predictor undergoing injection can take advantage of the injected knowledge with regard to the domain and the task that the predictor was trained for. Thus, for instance, if a knowledge base states that loans should be guaranteed to people from a given minority as long as annual income overcomes a given threshold, then any predictor undergoing injection of that knowledge base should output predictions respecting that statement or at least minimise violations with regard to it.

Purpose and benefits. One may be interested in performing SKI to reach a higher degree of control on what a sub-symbolic predictor is learning. In fact, SKI may either incentivise the predictor to learn some desirable behaviour or discourage it from learning some undesired behaviour. However, one may also exploit SKI to perform sub-symbolic or fuzzy manipulations of symbolic knowledge that would be otherwise unfeasible or hard to formalise via crisp symbols. While the latter option is further analysed by a number of authors (e.g.,[1, 30]), in the remainder of this section we focus on the former use case as it is better suited to serve the purposes of XAI.

Within the scope of XAI, SKI is a remarkable capability as it provides a workaround for the issues arising from the opacity of ML predictors. While SKE aims at reducing the opacity of a predictor by letting users understand its behaviour, SKI aims at bypassing the need for transparency. Indeed, predictors undergoing the injection of *trusted* symbolic knowledge provide higher guarantees about their behaviour, which will be more predictable and comprehensible.

Accordingly, the application of SKI to XAI brings a number of relevant opportunities, e.g., by letting human designers (i) endow sub-symbolic predictors with their common sense and, therefore, (ii) allowing them to finely control what predictors are learning, in particular, (iii) letting predictors learn about relevant situations despite poor data being available to describe them. Provided that adequate SKI procedures exist, all such use cases come at the price of handcrafting *ad hoc* knowledge bases reifying the designers' common sense in symbols and then injecting it into ordinary ML predictors.

3.2 Review Methodology

The overall review workflow is inspired by the goal question metric approach by the authors of [11]. In short, the workflow requires some clear research *goal(s)* to be fixed and then decomposed into a number of research questions the survey will then provide answers to. To produce such answers, the workflow requires scientific papers to be selected and analysed. To serve this purpose, the workflow requires a pool of *queries* to be identified. Such queries must be performed on the most relevant bibliographic search engines (e.g., Google Scholar, Scopus). Finally, the workflow requires the query results to be selected (or excluded) for further analyses following a reproducible criterion. Any subsequent analysis is then devoted to answering the aforementioned research questions, hence, drawing useful classifications and general conclusions.

For the sake of reproducibility, in the remainder of this subsection we delve into the details of how our SLR on symbolic knowledge extraction and injection is conducted.

We start by defining three different research goals (**Gs**):

G1 – *Understanding which are the features of SKE algorithms*
G2 – *Understanding which are the features of SKI algorithms*
G3 – *Probing the current level of technological readiness of SKE/SKI technologies*

Then, we break them down into the following research questions (**RQs**):

RQ1 (from **G1**) – *Which sort of ML predictors can SKE be applied to?*
RQ2 (from **G1**) – *Is there any requirement on the input data for SKE?*
RQ3 (from **G1**) – *Which kind of SK can be extracted from ML predictors?*
RQ4 (from **G1**) – *For which kind of AI tasks can SKE be exploited?*
RQ5 (from **G1**) – *How does SKE work?*
RQ6 (from **G2**) – *Which sorts of ML predictors can SKI be applied to?*
RQ7 (from **G2**) – *Which kind of SK can be injected into ML predictors?*
RQ8 (from **G2**) – *For which kind of AI tasks can SKI be exploited?*
RQ9 (from **G2**) – *How does SKI work?*
RQ10 (from **G3**) – *Which and how many SKE/SKI algorithms come with runnable software implementations?*

Note that research questions about SKE are analogous to those about SKI. In both cases, research questions are devoted to clarifying which kind of information SKE (resp., SKI) methods can accept as input (resp., produce as output), how they work, which AI tasks they can be used for (e.g., regression, classification), and which ML predictors they can be applied to (e.g., NN, SVM, etc.).

In order to answer the research questions above, we identify a number of queries to be performed on widely available bibliographic search engines. Queries involve the following keywords:

- (‘rule extraction’ ∨ ‘knowledge extraction’) ∧ (‘neural networks’ ∨ ‘support vector machines’)
- (‘pedagogical’ ∨ ‘decompositional’ ∨ ‘eclectic’) ∧ (‘rule extraction’ ∨ ‘knowledge extraction’)
- ‘symbolic knowledge’ ∧ (‘deep learning’ ∨ ‘machine learning’)
- ‘embedding’ ∧ (‘knowledge graphs’ ∨ ‘logic rules’ ∨ ‘symbolic knowledge’)
- ‘neural’ ∧ ‘inductive logic programming’

As far as bibliographic search engines are concerned, we exploit Google Scholar,[1] Scopus,[2] Springer Link,[3] ACM Digital Library,[4] and DBLP.[5]

For each search engine and query pair, we consider the first two pages of results. For each result, we inspect the title, abstract, and, in the case of ambiguity, the introduction, while trying and classifying it according to three disjoint circumstances: (i) the paper is a *primary work* describing some SKE or SKI method matching the broad definitions from 3.1, (ii) the paper is a *secondary work* surveying some portion of literature overlapping SKE or SKI (or both), and (iii) the paper is *unrelated* with regard to both SKE and SKI, hence, it is not relevant for this survey. Notably, secondary works selected in step 3.2 are valuable sources of primary works; hence, we recursively explored their bibliographies to further select other primary works. In particular, in this phase we leverage relevant secondary works such as [4, 7, 12, 18, 24, 25, 27, 53, 54, 56, 60], which we acknowledge as noteworthy (even though less extensive) surveys in the field of SKE or SKI.

We select 249 primary works, of which 132 works concern SKE and 117 concern SKI. We then analyse each primary work individually in order to provide answers to the aforementioned research questions. While doing so, we construct bottom-up taxonomies for both SKE and SKI.

Finally, we inspect each primary work to assess its technological status. We look for runnable software implementations corresponding to the method described in the primary work. In the case in which no software tool is clearly mentioned in the primary work or if the software is not technically accessible (e.g., website or repository is private or non-reachable) at the time of

[1] https://scholar.google.com
[2] https://www.scopus.com
[3] https://link.springer.com
[4] https://dl.acm.org
[5] https://dblp.uni-trier.de

the survey, we consider the method as lacking software implementations. Otherwise, we further distinguish among methods with reusable software libraries and methods with experimental code. In the first case, the software is ready for reuse either because it is published on public software repositories such as PyPi or because it is structured in such a way as to let users exploit it for custom purposes. If the software is tailored on the experiments mentioned in the primary work, then we consider it experimental.

4 SURVEY RESULTS

This section summarises the results of our survey. Answers for the research questions outlined in Section 3.2 are provided here.

We group research questions according to their main focus (SKE or SKI) and answer each question individually—grouping answers, when convenient, for the sake of conciseness. Answers consist of brief statistical reports showing the distribution of the surveyed SKE/SKI methods with regard to the dimension of interest for either SKE or SKI. Interesting dimensions are presented on the fly as part of our answers. This is deliberate since we select as 'interesting dimension' any relevant way of clustering the surveyed methods. We let taxonomies emerge from the literature rather than super-imposing any particular view of ours.

4.1 Symbolic Knowledge Extraction

By building upon secondary works, such as the work by the authors of [12] and the survey by the authors of [4], we identify three relevant dimensions by which SKE methods can be categorised: (i) the learning task(s) they support; (ii) the method's translucency; and (iii) the shape of the extracted knowledge. By analysing the surveyed SKE methods, we find these categories to be adequate. However, we identify new dimensions: (iv) the sort of input data the predictor undergoing extraction is trained upon and (v) the expressiveness of the extracted knowledge. In what follows, we answer research questions **RQ1** to **RQ5** and **RQ10** by focusing on these dimensions individually. Conversely, in the supplementary materials, we provide an overview of the 132 methods selected for SKE.

*4.1.1 **RQ1**: Which sort of ML predictors can SKE be applied to? **RQ5**: How does SKE work?* Answers for questions **RQ1** and **RQ5** are deeply entangled, as they are both related to SKE methods' *translucency*. Translucency deals with the need for SKE methods to inspect the internal structure of the underlying black-box model while producing the extracted rules.

SKE methods provide for translucency in two ways [4] and can be labelled accordingly as

decompositional if the method needs to inspect (even partially) the internal parameters of the underlying black-box predictor, e.g., neuron biases or connection weights for NNs, or support vectors for SVMs;

pedagogical if the algorithm *does not* need to take into account any internal parameter, but it can extract symbolic knowledge by only relying on the predictor's outputs.

Along this line, we observe that surveyed SKE methods can be grouped into as many big clusters depending on how they treat the predictor undergoing extraction.

With regard to **RQ1**, it is worth highlighting that pedagogical methods can be applied to any sort of supervised ML predictor, in principle despite the fact that the literature may only report particular cases of application to specific predictors. Conversely, each decompositional method focuses on a specific sort of supervised ML predictor. Hence, decompositional SKE methods can be further categorised with regard to which sort of supervised ML predictors they are tailored to. As detailed in Figure 2, the translucency is far from uniform for SKE methods. Indeed, nearly

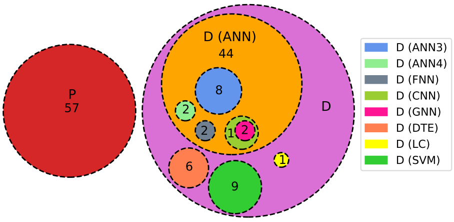


Fig. 2. Venn diagram categorising SKE methods with regard to *translucency*: pedagogical (P) or decompositional (D). For decompositional methods, we report the target predictor type: ANN⟨n⟩ = artificial NN (possibly having exactly ⟨n⟩ layers), CNN = convolutional NN, GNN = graph NN, FNN = fuzzy NN, SVM = support vector machines, DTE = decision tree ensembles, LC = linear classifiers.

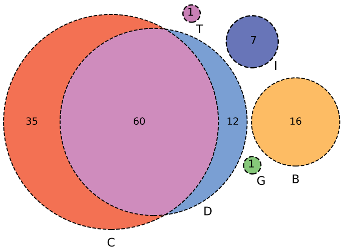


Fig. 3. Venn diagram categorising SKE methods with regard to the *input data type* required by the underlying predictor: binary (B), discrete (D), continuous (C), images (I), text (T), graphs (G).

half of the surveyed methods are pedagogical, whereas the rest are tailored to feed-forward NNs (possibly with fixed amounts of layers), SVM, linear classifiers, or decision tree ensembles.

With regard to **RQ5**, it is worth highlighting that pedagogical methods treat the underlying predictor as an *oracle* to be queried for predictions the symbolic knowledge shall emulate. Conversely, decompositional methods must look into the internal structure of predictors, hoping to detect meaningful patterns. For instance, SKE methods focusing on NNs may try to interpret inner neurons as meaningful expressions combining their ingoing synapses.

4.1.2 ***RQ2****: Is there any requirement on the input data for SKE?.* This question can be answered by looking at the accepted input data type of the surveyed SKE methods. In most cases, data is structured, i.e., it consists of tables of numberswith three different types of features:

Binary The feature can assume only two values, generally encoded with 0 and 1 (or −1 and 1, or `true` and `false`)
Discrete The feature can assume values drawn from a *finite* set of admissible values; notably, when this is the case, data science identifies two relevant sub-sorts of features: **ordinal** if the set of admissible values is *ordered* (hence, enabling the representation of the feature via some range of integer numbers) or **categorical** if that set is *un*ordered (hence, enabling the representation of the feature via one-hot encoding)
Continuous The feature can assume any real numeric value

Alternatively, data may consist of the following.

Images Matrices of pixels, possibly with multiple channels
Text Sequences of characters of arbitrary length
Graphs Data structures of variable sizes, consisting of nodes/vertices interconnected by edges/arcs

In Figure 3, we report absolute occurrence of the types of input features accepted by the surveyed SKE methods, as described by their authors.

As the reader may notice, the vast majority of surveyed methods are tailored to structured data with *continuous* and/or *discrete* features.

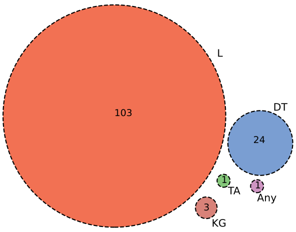


Fig. 4. Venn diagram categorising SKE methods with regard to the output knowledge's *shape*: rule lists (L), decision trees (DT) or tables (TA), knowledge graphs (KG).

4.1.3 RQ3: Which kind of SK can be extracted from ML predictors? Broadly speaking, any extracted SK should mirror (i.e., mimic) the operation of the ML predictor it has been extracted from. For *supervised* ML, this means that the extracted knowledge should express a *function*, mapping input features into output features (e.g., classes for classification tasks). Functions can be represented in symbols in several ways. Indeed, the SK extracted by the surveyed methods comes in various forms.

These forms can be categorised under both *syntactic* or *semantic* perspective. Here, *syntax* refers to the *shape* of the extracted SK, whereas *semantic* refers to what kind of logic formalism the extracted knowledge may leverage—which is a matter of *expressiveness*.

Shape of the extracted knowledge. As far as syntax is concerned, decision *rules* [19, 28, 40] and *trees* [9, 43] are the most widespread human-comprehensible formats for the output knowledge. Thus, the vast majority of surveyed methods adopt one of these. However, other solutions have been exploited as well—e.g., decision *tables*. In all cases, however, a common trait is that functions of real numbers are expressed by using *symbols* to denote the same input and output features the underlying ML predictor was trained on.

With regard to surveyed SKE methods, we identify four major admissible shapes:

Lists of rules Sequences of logic rules to be read in some predefined order

Decision trees See Section 2.1.2

Decision tables Concise visual rule representations specifying one or more conclusions for each set of different conditions. They can be *exhaustive* if all the possible combinations are listed or *incomplete* otherwise. Generally speaking, decision tables are structured as follows: there is a column (row) for each input and output variable and a row (column) for each rule. Each cell c_{ij} (c_{ji}) contains the value of the j-th variable for the i-th rule. An example of a decision table is provided in the supplementary material.

Knowledge graphs See Section 2.2.2.

Figure 4 sums up the occurrence of the different shapes of output rules required for SKE algorithms. As the reader may notice, the majority of the surveyed methods target rule *lists*. Arguably, this trend may be motivated by the great simplicity of rule lists in terms of readability and their algorithmic tractability.

Expressiveness of the extracted knowledge. Despite the fact that the extracted knowledge may contain statements of different shapes (e.g., rules, trees, tables), the readability, conciseness, and tractability of the extracted rules heavily depend on what those statements can contain—which, in turn, dictates what can (or cannot) be expressed. Generally, statements may contain *predicates* or *relations* among the symbols representing input or output features. These may (or may not) contain logic connectives as well as arithmetic or logic comparators. SKE methods can be categorised with regard to which and how many ways of combining symbols are admissible within statements.

Along this line, we identify five major formats for statements in the surveyed SKE methods.

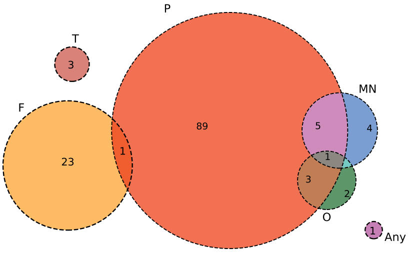


Fig. 5. Venn diagram categorising SKE methods with regard to the output knowledge's *expressiveness*: propositional (P), M-of-N (MN), fuzzy (F), or oblique (O) rules; or triplets (T).

Propositional rules are the simplest format, where statements consist of *propositions*, i.e., symbols denoting *Boolean* input/output features possibly interconnected via logic connectives (negation, conjunction, disjunction, etc.). Notice that statements containing relations (e.g., arithmetic comparisons) among *single*, *continuous* features and *constant* values are propositional as well.

Fuzzy rules are propositional rules where the truth value of conditions and conclusions are not limited to 0 and 1; rather, they can assume any value $\in [0,\ 1]$.

Oblique rules have conditions expressed as inequalities involving linear combinations of the input variables. This is different from the propositional case, as features may be compared to other features (rather than constants alone).

***m*-of-*n* rules** are particular types of rules where *Boolean* statements are grouped by n and each rule is *true* only if at least m literals (out of n) are *true*, with $m \leq n$. Notice that m-of-$(X_1, \ldots, X_n)$ is just a concise way of writing the disjunction among the conjunction of all possible m-sized combinations of n Boolean literals $X_1, \ldots, X_n$. Hence, m-of-n rules are just a concise way of writing rules of other types: if $X_1, \ldots, X_n$ are all predicative statements, then the expression m-of-$(X_1, \ldots, X_n)$ is predicative as well—and the same is true if $X_1, \ldots, X_n$ are oblique statements.

Triplets See Section 2.2.2.

Figure 5 summarises the occurrence of the different SK formats produced by the surveyed SKE algorithms. As the reader may notice, the vast majority of surveyed SKE methods produce predicative rules, i.e., rules composed of several Boolean statements about individual input features possibly interconnected via logic connectives. Arguably, this trend may be motivated by the great tractability of propositional rules and by their simplicity. In fact, to construct propositional rules, SKE algorithms may follow a *divide-et-impera* approach by focusing on one single input feature at a time—hence, enabling the simplification of the extraction process itself.

*4.1.4 **RQ4**: For which kind of AI tasks can SKE be exploited?* ML methods are commonly exploited in AI to serve specific purposes, e.g., classification, regression, and clustering. Regardless of the particular means by which SKE is attained, extraction aids the human users willing to inspect *how* those methods work. However, the particular AI tasks that ML predictors have been designed for play a pivotal role in determining what outputs users may expect from those predictors. A similar argument holds for extraction procedures, as the extracted knowledge should reflect the inner behaviour of the original predictor. Along this line, it is interesting to categorise SKE methods with regard to the AI task they assume for the ML predictors they are applied to.

Figure 6 summarises the occurrence of tasks among the surveyed SKE methods. Notably, most of them can be applied uniquely to *classifiers*, whereas a small portion of them is explicitly designed for *regressors*. Only a few methods can handle both categories.

In general, we observe how the surveyed methods are tailored to either classification or regression tasks—when not both. In either case, surveyed methods focus on supervised ML tasks. To

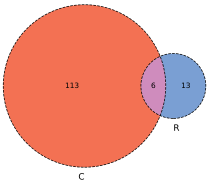


Fig. 6. Venn diagram categorising SKE methods with regard to the *targeted AI task*: classification (C) or regression (R).

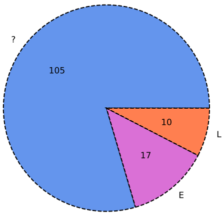


Fig. 7. Pie chart categorising SKE methods presence/lack of software implementations. 'L' denotes the presence of a reusable library, 'E' denotes experiment code, and '?' denotes lack of known technologies.

the best of our knowledge, currently, there are no SKE procedures tailored on unsupervised or reinforcement learning tasks.

4.1.5 **RQ10**: Which and how many SKE algorithms come with runnable software implementations?

Among the 132 surveyed methods for SKE, we found runnable software implementations for 27 (20.5%). Of these, 10 consist of reusable software libraries, whereas the others are just experimental code. Figure 7 summarises this situation. In the supplementary materials, we provide details about these implementations-including the algorithm that they implement and the link to the repository hosting the source code.

4.2 Symbolic Knowledge Injection

As far as SKI is concerned, we take into account no prior taxonomy. Despite the fact that the methods surveyed in this subsection come from well-studied (yet disjoint) research communities such as neuro-symbolic computation [7] and knowledge graph embedding [54], we are not aware of any prior work attempting to unify these research areas under the SKI umbrella.

Along this line, we cluster the surveyed SKI methods according to four orthogonal dimensions: (i) the type of SK they can inject, (ii) the strategy they follow to attain injection, (iii) the kind of predictors they can be applied to, and (iv) the aim they pursue while performing injection. In what follows, we answer research questions **RQ6** to **RQ10** by focusing on these dimensions individually. Conversely, in the supplementary materials, we overview the 117 methods selected for SKI.

4.2.1 **RQ7**: Which kind of SK can be injected into ML predictors?

Generally speaking, SKI methods support the injection of knowledge expressed by various formalisms despite each surveyed method focusing on some particular formalism. A key discriminating factor is whether the chosen formalism is machine interpretable or not other than human interpretable.

With regard to the formalism the input knowledge should adopt to support SKI, we may cluster the surveyed methods into two major groups:

Logic formulæ or **knowledge bases (KBs)** (i.e., sets of formulæ) adhering to either FOL or some of its subsets, which are therefore both machine and human interpretable. Here, admissible sub-categories reflect the kinds of logics described in Section 2.2.1. Ordered by decreasing expressiveness, these are:

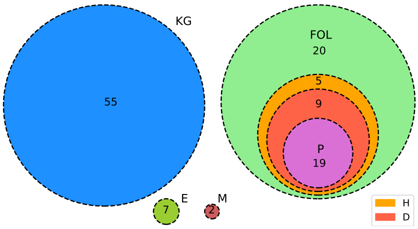


Fig. 8. Venn diagram categorising SKI methods with regard to the *input knowledge* type: knowledge graphs (KG), propositional logic (P), first-order logic (FOL), expert knowledge (E), Datalog (D), Horn logic (H), or modal logic (M).

Full first-order logic formulæ, including recursive terms, possibly containing variables, predicates of any arity, and logic connectives of any kind, possibly expressing definitions;

Horn logic (a.k.a. **Prolog**-like) where knowledge bases consist of head–body rules, involving predicates and terms of any kind

Datalog i.e., Horn clauses without recursive terms (only constant or variable terms allowed)

Modal logics i.e., extensions of some logic above with modal operators (e.g., □ and ◇), denoting the *modality* in which statements are true (e.g., *when*, in temporal logic)

Knowledge graphs i.e., a particular application of description logics aimed at representing entity–relation graphs

propositional logic where expressions are simply expressions involving Boolean variables and logic connectives

Expert knowledge i.e., any piece of human (but not necessarily machine) interpretable knowledge by which data generation can be attained. This might be the case of physics formulae, syntactical knowledge, or any form of knowledge that is usually held by a set of human experts, and, as such, is only accessible to human beings. For this reason, expert knowledge injection requires some data to be generated to reify its information in tensorial form. Of course, expert knowledge may be cumbersome to extract and requires human engineers to take care of data generation before any injection can occur.

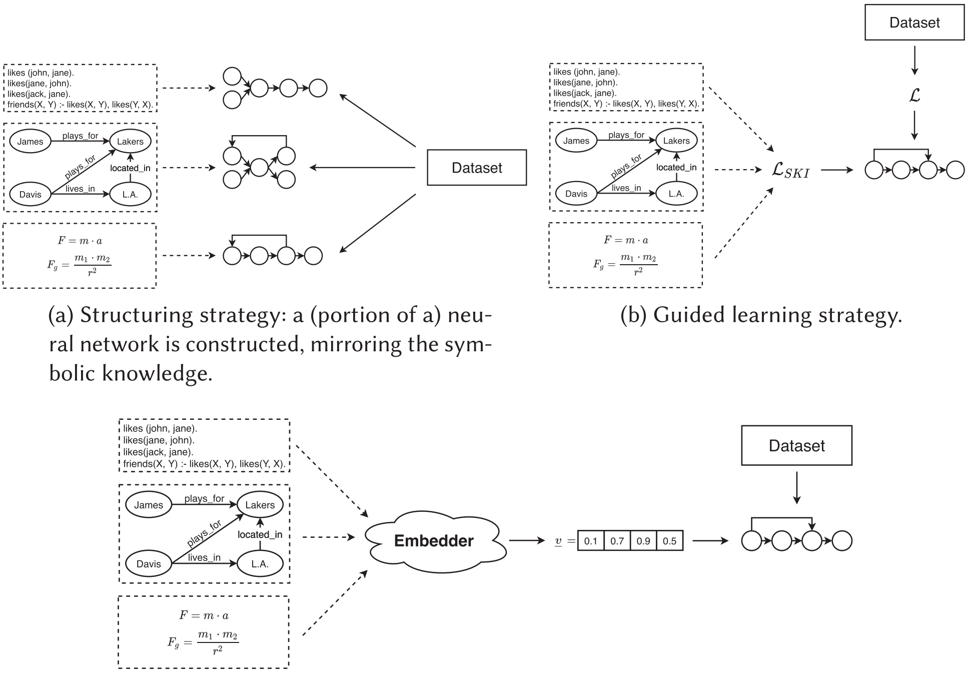
In Figure 8, we categorise the surveyed SKI methods with regard to their formalism of choice. Here, KGs are the most prominent cluster (including almost half of the surveyed methods), whereas model logic is the smallest. Methods tailored to FOL or its subsets (apart from KGs) form another relevant cluster. Among the FOL subsets, propositional logic plays a pivotal role, as it involves the relative majority of methods.

As long as the logic formalism is concerned, we consider and report the *actual* logic used in the papers. This is rarely explicitly stated by the authors in their papers. Thus, we deduce the actual logic used by each SKI method from the constraints that its logic is subject to according to its authors.

4.2.2 *RQ9: How does SKI work?* By analysing the surveyed SKI methods, we acknowledge great variety in the way that injection is performed. Arguably, however, such variety can be tackled by focusing on three major *strategies*, depicted in Figure 9 and summarised below:

Predictor structuring in which (a part of) a sub-symbolic predictor (commonly, NN) is created to mirror the symbolic knowledge via its own internal structure. A predictor is created or extended to mimic the behaviour of the SK to be injected. For instance, when it comes to NNs, their internal structure is crafted to represent logic predicates via neurons, and logic connectives via synapses.

Knowledge embedding in which SK is converted into numeric-array form—e.g., vectors, matrices, and tensors—to be provided as ‘ordinary’ input for the sub-symbolic predictor undergoing injection. Numeric data is generated out of symbolic knowledge. Any numeric



(a) Structuring strategy: a (portion of a) neural network is constructed, mirroring the symbolic knowledge.

(b) Guided learning strategy.

(c) Embedding strategy: the symbolic knowledge is converted in tensorial form and ML predictors are fed “as usual”.

Fig. 9. Overview of major strategies followed by surveyed SKI methods.

representation of this type is called *embedding* [of the original symbolic knowledge]. For example, this is the common strategy exploited by the knowledge graph embedding community [54] as well as by graph NNs [1, 30].

Guided learning (i.e., **constraining**) in which SK is used to steer the learning process of ML predictors by either penalising inconsistent behaviours or by incentivising consistent behaviours with regard to the SK. When the predictor undergoing injection is trained via an optimisation process involving loss functions being minimised (e.g., NN), guided learning is achieved by altering those loss functions in such a way that violations with regard to the SK increase the loss. A dual statement holds for predictors requiring training to step through maximization processes. A useful overview of these kinds of methods can be found in [22].

Figure 10 summarises the frequency of these strategies among the surveyed SKI algorithms. Notably, the distribution of surveyed SKI methods among the three categories above is quite balanced.

4.2.3 ***RQ6**: Which kinds of ML predictors can SKI be applied to?* Virtually *all* surveyed SKI methods are designed to inject knowledge into NNs. However, as this survey spans over 2 decades, the kinds of NNs supported by SKI methods are manifold despite the fact that each method is tailored to specific kinds of NNs.

Accordingly, surveyed SKI methods can be classified with regard to the particular kind of NN they support. As detailed in Figure 11, admissible choices along this line fit the many kinds of NN discussed in Section 2.1.2, as follows.

Feed-forward NNs multi-layered NNs in which neurons from layer i are only connected with layer $i + 1$, and multiple (≥ 2) layers may exist

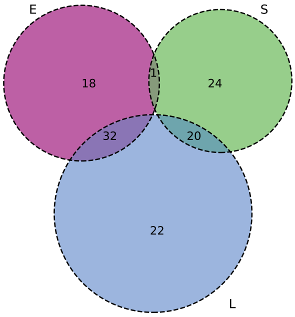


Fig. 10. Venn diagram categorising SKI methods with regard to *strategy*: structuring (S), embedding (E), or guided learning (L).

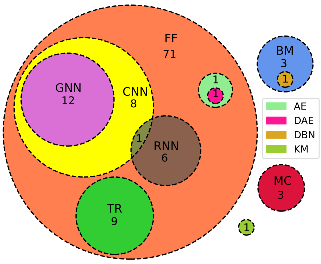


Fig. 11. Venn diagram categorising SKI methods with regard to the *targeted predictor* type: feed-forward (FF), convolutional (CNN), graph (GNN) or recurrent (RNN) neural networks, Boltzmann machines (BM), Markov chains (MC), transformers (TR), auto-encoders (AE), deep belief networks (DBN), denoising auto-encoders (DAE), kernel machines (KM).

Convolutional NNs particular cases of feed-forward NNs, involving convolutional layers as well
Graph NNs particular cases of convolutional NNs tailored on graph-like data
Recurrent NNs particular cases of NNs admitting loops among layers
Boltzmann machine a particular neural architecture in which connections are undirected, i.e., every node is connected to every other node
Transformer particular case of NN that leverages a self-attention mechanism, i.e., differentially weighting parts of the input data depending on their significance
Auto-encoders particular cases of feed-forward NNs, characterised by a bottleneck architecture used to learn reduced data encodings through learning to regenerate the input from the encoding
Deep belief network a composition of multiple Boltzmann machines, stacked together, in a feed-forward fashion
Denoising auto-encoder particular case of auto-encoder working over corrupted input

Notable exceptions are as follows.

Kernel machines ML models relying on kernels, i.e., similarity measures between observed patterns;
Markov chains state machines with probabilities on state transitions, modelling stochastic phenomena

The reason why the vast majority of methods rely on (some sort of) NN is straightforward: methods tailored to GNNs (resp., CNNs) assume the networks to accept specific kinds of data as input, e.g., graphs (resp., images), while ordinary feed-forward NNs accept raw vectors of real numbers.

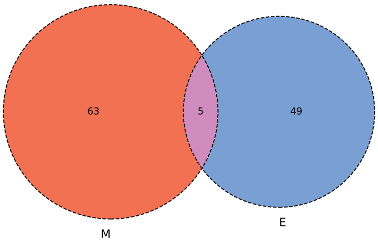


Fig. 12. Venn diagram categorising SKI methods with regard to *aim*: knowledge manipulation (M) or enrich (E).

*4.2.4 **RQ8**: For which kind of AI tasks can SKI be exploited?* Unlike SKE methods, which uniquely serve the purpose of inspecting black-box predictors by mimicking the way they address supervised learning tasks, SKI methods from the literature may serve multiple purposes. In Figure 12, we identify the following two major purposes that SKI methods may pursue by targeting symbolic or sub-symbolic AI tasks.

Symbolic knowledge manipulation in which SKI enables the *sub*-symbolic manipulation of *symbolic* knowledge by letting sub-symbolic predictors treat SK similarly to what is done by symbolic engines. In doing so, SKI supports symbolic-AI tasks such as

logic inference in its many forms (e.g., deductive, inductive, and probabilistic), i.e., drawing conclusions out of symbolic KB

information retrieval looking for information in symbolic KB

KB completion finding (and adding) missing information in symbolic KB

KB fusion merging several KBs into a single one, taking care of (possibly, syntactically different) overlaps

The key point here is supporting tasks in which both inputs and outputs are symbolic in nature, but leveraging sub-symbolic methods to gain speed, fuzziness, and robustness against noise.

Learning support (i.e., **enrich**) in which SKI lets *sub*-symbolic methods consume *symbolic* knowledge to either improve or enrich learning capabilities. In doing so, SKI supports ordinary ML tasks such as classification by allowing ML predictors to process (or take advantage of) structured symbolic knowledge. The underlying idea of such approaches is that there exist some concepts that are cumbersome or troublesome to learn from examples—e.g., syntactical concepts and semantics. Therefore, SK expressing these high-level concepts may be injected directly into the model to be trained.

As the reader may note from the picture, surveyed SKI methods are quite balanced with regard to the categories above, with a slight preference for SK manipulation.

*4.2.5 **RQ10**: Which and how many SKI algorithms come with runnable software implementations?* Among the 117 surveyed methods for SKI, we found runnable software implementations for 60 (51.3%). Of these, 11 consist of reusable software libraries, whereas the others are just experimental code. Figure 13 summarises this situation. In the supplementary materials, we provide details about these implementations, including the algorithm they implement and the link to the repository hosting the source code.

5 DISCUSSION

Figure 14 summarises the main contribution of our article: the taxonomies for SKE and SKI that we induced from the surveyed literature. Generally speaking, such taxonomies are useful tools to categorise present (and, hopefully, future) SKE/SKI methods and to highlight the relevant features of each particular method. In this way, the interested readers may figure out what to expect from

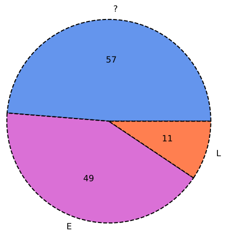


Fig. 13. Pie chart categorising SKI methods presence/lack of software implementations. Here, 'L' denotes the presence of a reusable library, 'E' denotes experiment code, and '?' denotes lack of known technologies.

any given SKE/SKI method as well as perform general analyses concerning the state-of-the-art. Accordingly, in this section we analyse our taxonomies, elaborating on the current challenges and future perspectives.

It is worth mentioning that our taxonomies involve both 'stable' and 'contingent' categories by which SKE/SKI methods can be described. These are represented as either white or grey boxes in Figure 14. Stable categories are time-independent and they are not susceptible to change in the near future, whereas contingent categories are subject to trends and may evolve. Consider, for instance, SKE methods (see Figure 14), categorised with regard to their output knowledge. While expressiveness is a stable sub-category, its actual sub-sub-categories are contingent, meaning that new ones may be added in the future.

5.1 SKE Taxonomy

As shown in Figure 14, SKE methods can be classified by (i) translucency, (ii) targeted AI task, (iii) nature of the input data, and (iv) form of the output knowledge. With regard to Section 5.1, SKE methods can either be categorised as pedagogical or decompositional. In the particular case of decompositional methods, the actual targeted predictor is also relevant; possibilities *currently* include NNs, DTs, SVMs, and linear classifiers. With regard to Section 5.1, SKE methods may target classification or regression tasks, or both. In any case, they *currently* target supervised ML tasks alone. With regard to Section 5.1, SKE methods accept predictors trained upon binary, discrete, or continuous data, as well as images, graphs, and text. Finally, with regard to Section 5.1, SKE methods may produce symbolic knowledge of different shapes and with different expressiveness. Shapes may *currently* involve rule lists as well as graphs, decision trees, or tables. Conversely, as long as expressiveness is involved, symbolic knowledge may be propositional or fuzzy, possibly including M-of-N-like statements, or may be expressed as triplets or oblique rules.

About translucency. It is worth stressing the relevance of pedagogical methods from the engineering perspective. If properly implemented, pedagogical methods may be exploited in combination with predictors of any kind. Of course, they are expected to reach lower performances with regard to decompositional ones, as they access less information. On the other side, decompositional methods may be more precise at the expense of generality.

About input data. We recall that binary features are particular cases of discrete features, whereas discrete features are, in turn, particular cases of continuous features. Hence, it is worthwhile noticing that extractors requiring only binary features can be applied to categorical datasets by preprocessing discrete attributes via **one-hot encoding (OHE)**. Analogously, extractors requiring discrete features can work with continuous attributes if those continuous features are *discretised*. Finally, continuous features can be converted into binary ones by performing discretisation and OHE in that order.

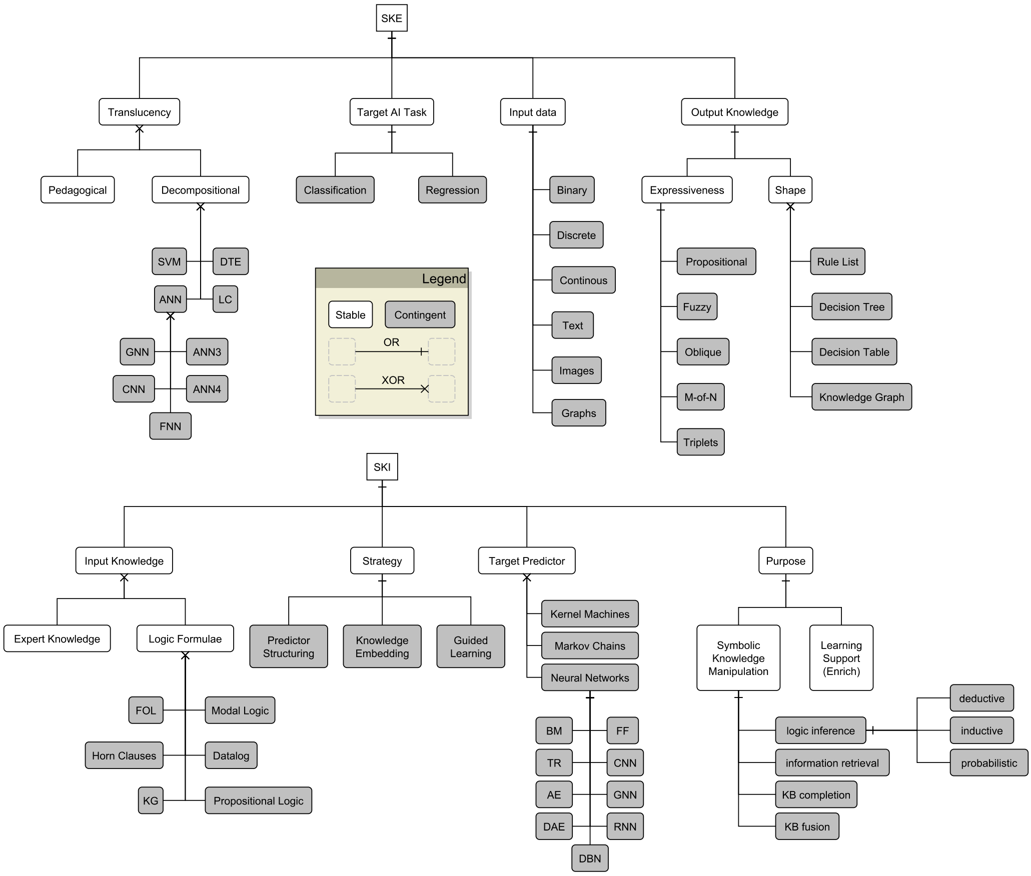


Fig. 14. Summary of SKE and SKI taxonomies derived from the literature, as discussed in Section 4.

While these transformations can always be applied in the general case, some authors have included them in their SKE methods at the design level. Hence, some papers explicitly count discretisation or OHE as part of the SKE methods they propose. This is the case, for instance, of the methods enclosed in the intersection between the ‘C’ and ‘D’ sets in Figure 3 (and labelled as ‘C+D’ in the supplementary materials). Other methods may instead rely upon other discretisation strategies, such as the ones surveyed by [58].

About output knowledge. It is worth stressing that differences among rule lists, decision trees, and tables are mostly syntactic, as conversions among these forms are possible in the general case (see the supplementary materials for examples). As far as expressiveness is concerned, we remark that all logic formalisms currently in use for SKE are essentially particular cases of propositional logic—possibly under a fuzzy interpretation. This implies that the full power of FOL is far from being fully exploited in practice.

Finally, we point out some correlations among the expressiveness of output rules and the nature of the predictor they are extracted from, as well as the input data it is trained on. For instance, SKE methods working with *continuous* input data are more likely to adopt oblique rules—or, at least, propositional rules with arithmetic comparisons. In fact, decisions are drawn by comparing numeric variables with constants or among each other. Another example: some *decompositional*

SKE methods focusing upon NN adopt M-of-N statements. Arguably, the reason is that M-of-N expressions aggregate several elementary statements into a single formula, similar to how neurons aggregate synapses from previous layers in NN. Hence, such methods approximate neurons via M-of-N expressions.

On SKE methods' chronology. In conclusion, we stress the chronological distribution of SKE methods. As highlighted by the supplementary materials, the majority of SKE methods have been proposed ranging from the 1990s to the 2010s. Contributions slowed down after that until the 2020s, when SKE gained new momentum.

In our opinion, research on ML interpretability gained momentum more than once in the history of AI. Each time sub-symbolic AI attracted the interest of researchers, so did the need to make it more comprehensible. Arguably, this is the reason why most SKE-related works are concentrated around the 2000s. We have been witnessing the novel spring of sub-symbolic AI [35], which is, in turn, motivating researchers' interest in XAI. Arguably, this is why SKE is gaining novel momentum in recent years.

5.2 SKI Taxonomy

As shown in Figure 14, SKI methods can be classified by (i) form of the input knowledge, (ii) followed strategy, (iii) targeted predictor type, and (iv) purpose. With regard to Section 5.2, SKI methods can either accept logic formulæ or expert knowledge as input. In the former case, *current* possibilities include FOL and its subsets, and in particular knowledge graphs. With regard to Section 5.2, SKI methods may *currently* follow one of three strategies: predictor structuring, knowledge embedding, or guided learning. With regard to Section 5.2, SKI methods *currently* mostly target NN-based predictors other than Markov chains and kernel machines. Finally, with regard to Section 5.2, SKI methods may pursue two kinds of purposes non-exclusively: manipulating symbolic knowledge or supporting/enriching learning. In the former case, *current* possibilities involve symbolic AI-related tasks such as logic inference (and its many forms), information retrieval, and KB completion/fusion.

About input knowledge and injection strategies. Logic formulæ are the most common approach to defining prior concepts to be injected. This is true in particular for SKI approaches following model structuring or guided learning strategies. Via logic formulæ, they express criteria that sub-symbolic models should satisfy or emulate. However, these types of methods often require formulæ to be grounded. Grounding introduces computational burden and hinders capability of representing recursive or infinite data structures—hence, limiting what can actually be injected.

Conversely, KGs are the most common knowledge representation approach when it comes to performing SKI following the knowledge embedding strategy. This is unsurprising, given that 'knowledge graph embedding' is a research line *per se*.

About target predictors. NNs play a pivotal role in SKI. Arguably, the reason lies in the great *malleability* of NNs with regard to their structure and training as well as their *flexibility* with regard to feature learning. In fact, NNs come in different shapes as different architectures may be constructed by connecting neurons in various ways. This is fundamental to supporting SKI via *predictor structuring*. Furthermore, as long as their architectures are DAGs, NNs can be trained via gradient descent, i.e., by minimising a loss function arbitrarily defined. This is, in turn, fundamental to supporting SKI via *guided learning*. Finally, feature learning is a characterising capability of NNs, making them capable of automatically eliciting the relevant aspects they should focus up with regard to input data. This is the reason why NNs are well suited for the knowledge embedding strategy as well. To the best of our knowledge, there exists no other type of predictor having similar flexibility and malleability.

On SKI methods' chronology. In conclusion, we stress the chronological distribution of SKI methods. As highlighted by the supplementary materials, the majority of SKI methods were proposed after 2010 and, notably, the amount of contribution has exploded since 2015.

In our opinion, this distribution is due to the composite effect of three major drivers: **natural language processing (NLP)**, XAI, and **neuro-symbolic computation (NSC)**. Arguably, all such drivers have been gaining momentum in the last few years, due to the success of ML and **deep learning (DL)**. NLP reached unprecedented performance levels after it started leveraging DL, possibly combined with KGs and the corresponding SKI methods. Similarly, a portion of XAI-related contributions proposed SKI methods aimed at controlling, constraining, or guiding what predictors learn from data. Finally, NSC has recently emerged as a field exploiting SKI methods to process logic knowledge sub-symbolically by exploiting the malleability of NNs.

5.3 Challenges

We observe that SKE algorithms focus exclusively on supervised learning tasks—i.e., classification and regression—while they do not tackle unsupervised or reinforcement learning tasks, e.g., clustering or optimal policy search. One may argue that clustering algorithms are not opaque—e.g., K-nearest neighbours—despite operating on numeric data. However, *pedagogical* SKE algorithms could be used on clustering predictors with no or minimal adjustments, as trained clustering predictors are essentially classifiers upon anonymous classes. Similarly, it could be possible to perform extraction on predictors trained using reinforcement learning with existing SKE algorithms. Future literature on SKE for unsupervised learning would be needed.

The vast majority of SKI algorithms accept knowledge in the form of KGs—i.e., description logic—or propositional logic (see Figure 8), which are much less expressive than FOL. These logics lack support for recursion and function symbols, meaning that the user is quite limited in providing knowledge to predictors. The reason is that common ML predictors are acyclic (e.g., NNs), meaning that there is no straightforward way to integrate recursion or indefinitely deep data structures without severe information loss due to approximations. Hence, future research efforts concerning SKI should consider addressing the injection of logics involving recursive clauses or arbitrarily deep data structures.

5.4 Opportunities

We propose a brief discussion on the benefits arising from the *joint* exploitation of both SKI and SKE in the engineering of AI solutions: (i) the possibility of *debugging* sub-symbolic predictors and (ii) the exploitation of symbolic knowledge as the *lingua franca* among heterogeneous hybrid systems. In the remainder of this sub-section, we delve into the details of these expected benefits.

5.4.1 Debugging Sub-symbolic Predictors.

Debugging is a common activity for software programmers: it aims at spotting and fixing *bugs* in computer programs under production/maintenance. A bug is some unknown error contained in the program that leads to an unexpected or undesired observable behaviour of the computer(s) running that program. The whole procedure relies on the underlying assumption that computer programs are intelligible to the programmer debugging them and that the program can be precisely edited to fix the bug.

One may consider XAI techniques as means of debugging sub-symbolic predictors. In this metaphor, sub-symbolic predictors correspond to computer programs despite the fact that they are not manually written by programmers but rather learned from data, whereas data scientists correspond to programmers. However, debugging sub-symbolic predictors is hard because of their

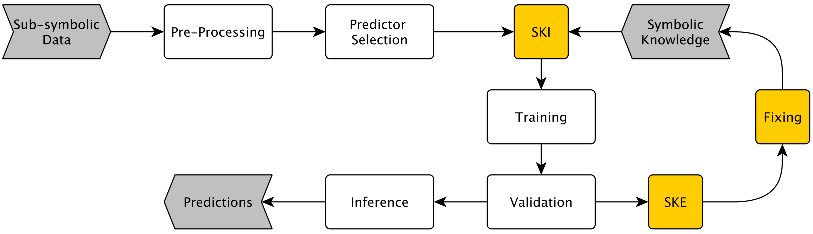


Fig. 15. ML workflow enriched with SKI and SKE phases. On the right, the train–extract–fix–inject loop is represented.

opacity, which makes their inner behaviour poorly intelligible for data scientists, and because they cannot be precisely edited after training and should be retrained from scratch instead. We discuss here the role of SKE and SKI in overcoming these issues, hence, allowing data scientists to debug sub-symbolic predictors.

Figure 15 provides an overview of how SKI and SKE fit the generic ML workflow. The figure stresses the relative position of both SKI and SKE with regard to the other phases of the ML workflow. Notably, SKI should occur before (or during) training, whereas SKE should occur after it. However, Figure 15 also stresses the addition of a *loop* into an otherwise linear workflow (right-hand side of the figure). We call it the **'train–extract–fix–inject' (TEFI)** loop, which we argue is a possible way to debug sub-symbolic predictors.

In the TEFI loop, SKE is the basic mechanism by which the inner operation of a sub-symbolic predictor (i.e., 'the program' in the metaphor) is made intelligible to data scientists. The extracted knowledge may then be understood by data scientists and debugged—looking for pieces of knowledge that are wrong with regard to data scientist expectations. Then, symbolic knowledge may be precisely edited and fixed. SKI is the basic mechanism by which a trained predictor is precisely edited to adhere to the fixed symbolic knowledge.

5.4.2 Symbolic Knowledge as the Lingua Franca *for Intelligent Systems.* Intelligent systems can be suitably modelled and described as composed of several intelligent, heterogeneous, and hybrid computational agents interoperating, and possibly communicating, among each other. Here, a computational agent is any software or robotic entity capable of computing other than perceiving and affecting some given environment—be it the Web, the physical world, or anything in between. To make the overall systems intelligent, these agents should be capable of a number of intelligent behaviours, ranging from image, speech, or text recognition to autonomous decision-making, planning, or deliberation. Behind the scenes, these agents may (also) leverage sub-symbolic predictors possibly trained on locally available data as well as symbolic reasoners, solvers, or planners to support these kinds of intelligent behaviours. Such agents are *hybrid*, meaning that they involve both symbolic and sub-symbolic AI facilities. However, interoperability may easily be a mirage because of (i) the wide variety of algorithms, libraries, and platforms supporting sub-symbolic ML other than (ii) the possibly different data items each agent may locally collect and later train predictors on. Each agent may learn (slightly) different behaviours due to the differences in the training data and in the actual ML workflow it adopts locally. When this is the case, exchange of behavioural knowledge may become cumbersome or infeasible.

In such scenarios, SKI and SKE may be enablers of a higher degree of interoperability, by supporting the exploitation of symbolic knowledge as the *lingua franca* for heterogeneous agents. Hybrid agents may exploit SKE to extract symbolic knowledge out of their local sub-symbolic predictors and exchange (and possibly improve) that symbolic knowledge with other agents. Then, any possible improvement of the symbolic knowledge attained via interaction may be back-propagated into local sub-symbolic predictors via SKI, enabling agents' behaviour to improve as well.

5.5 Limitations

This SLR means to be as comprehensive, precise, and reproducible as possible. Nonetheless, we acknowledge two potential limitations: *(i)* the expected life span of our taxonomies and *(ii)* terminology issues in the literature.

Both SKE and SKI are becoming increasingly popular topics; further advancements have to be expected for the next decade, at least. Hence, our taxonomies may require to be verified and possibly updated sometime in the future. The straightforward methodological approach defined by our SLR, however, should ensure a clear path to future reproductions of this work.

Also, an evolution in the naming conventions clearly emerge from our analysis. Through the years, SKE has been referred to in disparate ways—e.g., "rule extraction" [4] or "knowledge distillation" [57], to name just two. The same holds for SKI: its naming conventions are commonly based on the injection strategy, yet they rarely contain the word *injection*. Thus, we may have missed some works while collecting papers simply because they were using different naming conventions that we were not able to discover. This is an inherent issue of the keyword-based methodology we adopted for SLR. To minimise issues in the classifications of present and future SKE/SKI methods, we provide loose definitions and carefully read papers to determine whether they match our definitions or not. However, the existence of missing works for unexpected terminology choices cannot be excluded.

6 CONCLUSION

In this article, we survey the state-of-the-art of symbolic knowledge extraction and injection under an XAI perspective. Stemming from two original definitions, we *systematically* explore the literature of both SKE and SKI, spanning a period of 4 decades. Our goal is to elicit the major characteristics of SKE/SKI algorithms from the literature (**G1** and **G2**), deriving general taxonomies that we hope other researchers may exploit. Another goal is to assess the current state of technologies (**G3**) by identifying software implementations of SKE/SKI techniques.

Considerable efforts were spent in keeping our review *reproducible* as prescribed by the goal question metric approach in [11]. Along this line, we design 10 research questions (**RQ1**–**RQ10**), and we engineer *ad hoc* queries to be performed on most relevant search engines for scientific literature. We select 249 primary works, almost evenly distributed among SKE and SKI, along with 11 secondary works. By analysing these papers, we define and discuss two general taxonomies for both SKE and SKI, which are general enough to categorise present (and possibly future) methods.

Roughly, surveyed methods are categorised with regard to what they accept as input and produce as output (in terms of symbolic knowledge or predictors), along with how they operate and why. We also collect data about which and how many SKE/SKI methods come with runnable software implementations (87, i.e., 34.9%). In the supplementary materials, we also provide Web homepages for the available implementations.

Overall, the implications of our study are manifold. It demonstrates how SKE and SKI are currently hot topics of AI research. The literature already contains hundreds of contributions and our taxonomies provide an effective tool for navigating it. Hopefully, our SLR can also serve as a map for future contributions, which we expect to flourish soon and abundantly. Our survey summarises what has already been done and what is currently lacking (see Section 5.4).

REFERENCES

[1] Andrea Agiollo, Giovanni Ciatto, and Andrea Omicini. 2021. Graph neural networks as the copula mundi between logic and machine learning: A roadmap. In *WOA 2021 – 22nd Workshop "From Objects to Agents"* (Bologna, Italy) *(CEUR*

Workshop Proceedings, Vol. 2963), Roberta Calegari, Giovanni Ciatto, Enrico Denti, Andrea Omicini, and Giovanni Sartor (Eds.). CEUR-WS.org, Bologna, Italy, 98–115. http://ceur-ws.org/Vol-2963/paper18.pdf

[2] Andrea Agiollo and Andrea Omicini. 2022. GNN2GNN: Graph neural networks to generate neural networks. In *Uncertainty in Artificial Intelligence (Proceedings of Machine Learning Research, Vol. 180)*. ML Research Press, 32–42. https://proceedings.mlr.press/v180/agiollo22a.html Proceedings of the 38th Conference on Uncertainty in Artificial Intelligence, UAI 2022, 1-5 August 2022, Eindhoven, The Netherlands.

[3] Mikós Ajtai and Yuri Gurevich. 1994. Datalog vs first-order logic. *J. Comput. System Sci.* 49, 3 (1994), 562–588. https://doi.org/10.1016/S0022-0000(05)80071-6

[4] Robert Andrews, Joachim Diederich, and Alan B. Tickle. 1995. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems* 8, 6 (1995), 373–389. https://doi.org/10.1016/0950-7051(96)81920-4

[5] Franz Baader. 2003. Basic description logics. In *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, USA, 43–95.

[6] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. 2020. Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion* 58, December 2019 (2020), 82–115. https://doi.org/10.1016/j.inffus.2019.12.012

[7] Tarek R. Besold, Artur S. d'Avila Garcez, Sebastian Bader, Howard Bowman, Pedro M. Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, Luís C. Lamb, Daniel Lowd, Priscila Machado Vieira Lima, Leo de Penning, Gadi Pinkas, Hoifung Poon, and Gerson Zaverucha. 2017. Neural-symbolic learning and reasoning: A survey and interpretation. *Neuro-Symbolic Artificial Intelligence: The State of the Art* 342 (2017), 1–51. https://doi.org/10.3233/FAIA210348 arXiv:1711.03902

[8] Ronald J. Brachman and Hector J. Levesque. 2004. The tradeoff between expressiveness and tractability. In *Knowledge Representation and Reasoning*, Ronald J. Brachman and Hector J. Levesque (Eds.). Morgan Kaufmann, San Francisco, 327–348. https://doi.org/10.1016/B978-155860932-7/50101-1

[9] Leo Breiman, Jerome Friedman, Charles J. Stone, and Richard A. Olshen. 1984. *Classification and Regression Trees*. CRC Press.

[10] Jenna Burrell. 2016. How the machine 'thinks': Understanding opacity in machine learning algorithms. *Big Data & Society* 3, 1 (2016), 12 pages. https://doi.org/10.1177/2053951715622512

[11] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. 1994. The goal question metric approach. In *Encyclopedia of Software Engineering*, John J. Marciniak (Ed.). John Wiley & Sons, Inc., 528–532.

[12] Roberta Calegari, Giovanni Ciatto, and Andrea Omicini. 2020. On the integration of symbolic and sub-symbolic techniques for XAI: A survey. *Intelligenza Artificiale* 14, 1 (2020), 7–32. https://doi.org/10.3233/IA-190036

[13] Davide Calvaresi, Giovanni Ciatto, Amro Najjar, Reyhan Aydoğan, Leon Van der Torre, Andrea Omicini, and Michael Schumacher. 2021. Expectation: Personalized explainable artificial intelligence for decentralized agents with heterogeneous knowledge. In *Explainable and Transparent AI and Multi-Agent Systems*, Davide Calvaresi, Amro Najjar, Michael Winikoff, and Kary Främling (Eds.). LNCS, Vol. 12688. Springer, Cham, 331–343. https://doi.org/10.1007/978-3-030-82017-6_20

[14] Giovanni Ciatto, Roberta Calegari, Andrea Omicini, and Davide Calvaresi. 2019. Towards XMAS: eXplainability through multi-agent systems. In *AI&IoT 2019 – Artificial Intelligence and Internet of Things 2019 (CEUR Workshop Proceedings, Vol. 2502)*, Claudio Savaglio, Giancarlo Fortino, Giovanni Ciatto, and Andrea Omicini (Eds.). CEUR-WS.org, Rende, CS, Italy, 40–53. http://ceur-ws.org/Vol-2502/paper3.pdf

[15] Giovanni Ciatto, Davide Calvaresi, Michael I. Schumacher, and Andrea Omicini. 2020. An abstract framework for agent-based explanations in AI. In *19th International Conference on Autonomous Agents and MultiAgent Systems* (Auckland, New Zealand), Amal El Fallah Seghrouchni, Gita Sukthankar, Bo An, and Neil Yorke-Smith (Eds.). IFAAMAS, Auckland, New Zealand, 1816–1818. http://ifaamas.org/Proceedings/aamas2020/pdfs/p1816.pdf

[16] Giovanni Ciatto, Michael I. Schumacher, Andrea Omicini, and Davide Calvaresi. 2020. Agent-based explanations in AI: Towards an abstract framework. In *Explainable, Transparent Autonomous Agents and Multi-Agent Systems*, Davide Calvaresi, Amro Najjar, Michael Winikoff, and Kary Främling (Eds.). LNCS, Vol. 12175. Springer, Cham, Auckland, New Zealand, 3–20. https://doi.org/10.1007/978-3-030-51924-7_1

[17] Philipp Cimiano. 2006. *Ontology Learning and Population from Text*. Springer US. https://doi.org/10.1007/978-0-387-39252-3

[18] Artur S. d'Avila Garcez, Krysia Broda, and Dov M. Gabbay. 2001. Symbolic knowledge extraction from trained neural networks: A sound approach. *Artificial Intelligence* 125, 1-2 (2001), 155–207. https://doi.org/10.1016/S0004-3702(00)00077-1

[19] Alex A. Freitas. 2014. Comprehensible classification models: A position paper. *ACM SIGKDD Explorations Newsletter* 15, 1 (June 2014), 1–10. https://doi.org/10.1145/2594473.2594475

[20] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. 2016. *Deep Learning*. MIT Press, Cambridge, MA, USA. http://www.deeplearningbook.org/

[21] Bryce Goodman and Seth Flaxman. 2017. European Union regulations on algorithmic decision-making and a "Right to Explanation". *AI Magazine* 38, 3 (2017), 50–57. https://doi.org/10.1609/aimag.v38i3.2741

[22] Marco Gori (Ed.). 2018. *Machine Learning: A Constraint Based Approach*. Morgan Kaufmann, Cambridge, MA, USA. 560 pages. https://doi.org/10.1016/C2015-0-00237-4

[23] C. Cordell Green and Bertram Raphael. 1968. The use of theorem-proving techniques in question-answering systems. In *Proceedings of the 23rd ACM National Conference (ACM 1968)*, Richard B. Blue Sr. and Arthur M. Rosenberg (Eds.). ACM, New York, NY, USA, 169–181. https://doi.org/10.1145/800186.810578

[24] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. 2018. A survey of methods for explaining black box models. *Comput. Surveys* 51, 5 (2018), 1–42. https://doi.org/10.1145/3236009

[25] Tameru Hailesilassie. 2016. Rule extraction algorithm for deep neural networks: A review. *CoRR* abs/1610.05267 (2016), 1–6. arXiv:1610.05267 http://arxiv.org/abs/1610.05267

[26] Alfred Horn. 1951. On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic* 16, 1 (1951), 14–21. https://doi.org/10.2307/2268661

[27] Johan Huysmans, Bart Baesens, and Jan Vanthienen. 2006. *Using Rule Extraction to Improve the Comprehensibility of Predictive Models*. Technical Report 0612. K. U. Leuven. https://doi.org/10.2139/ssrn.961358

[28] Johan Huysmans, Karel Dejaeger, Christophe Mues, Jan Vanthienen, and Bart Baesens. 2011. An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decision Support Systems* 51, 1 (2011), 141–154. https://doi.org/10.1016/j.dss.2010.12.003

[29] Sotiris B. Kotsiantis. 2007. Supervised machine learning: A review of classification techniques. *Informatica (Slovenia)* 31, 3 (2007), 249–268. http://www.informatica.si/index.php/informatica/article/view/148

[30] Luís C. Lamb, Artur S. d'Avila Garcez, Marco Gori, Marcelo O. R. Prates, Pedro H. C. Avelar, and Moshe Y. Vardi. 2020. Graph neural networks meet neural-symbolic computing: A survey and perspective. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence, IJCAI 2020*, Christian Bessiere (Ed.). ijcai.org, Yokohama, Japan, 4877–4884. https://doi.org/10.24963/ijcai.2020/679

[31] Hector J. Levesque and Ronald J. Brachman. 1987. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence* 3 (1987), 78–93. https://doi.org/10.1111/j.1467-8640.1987.tb00176.x

[32] Zachary C. Lipton. 2018. The mythos of model interpretability. *Queue* 16, 3 (June 2018), 31–57. https://doi.org/10.1145/3236386.3241340

[33] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. 2018. Progressive neural architecture search. In *Proceedings of the 15th European Conference on Computer Vision (ECCV 2018) – Part I (Lecture Notes in Computer Science, Vol. 11205)*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Springer, Munich, Germany, 19–35. https://doi.org/10.1007/978-3-030-01246-5_2

[34] John W. Lloyd (Ed.). 1990. *Computational Logic*. Springer, Berlin. https://doi.org/10.1007/978-3-642-76274-1

[35] Jocelyn Maclure. 2020. The new AI spring: A deflationary view. *AI & Society* 35, 3 (2020), 747–750. https://doi.org/10.1007/s00146-019-00912-z

[36] J. A. Makowsky. 1987. Why horn formulas matter in computer science: Initial structures and generic examples. *J. Comput. System Sci.* 34, 2 (April–June 1987), 266–292. https://doi.org/10.1016/0022-0000(87)90027-4

[37] George F. McNulty. 1977. Fragments of first order logic, I: Universal horn logic. *Journal of Symbolic Logic* 42, 2 (1977), 221–237. https://doi.org/10.2307/2272123

[38] Tim Miller. 2019. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence* 267 (2019), 1–38. https://doi.org/10.1016/j.artint.2018.07.007

[39] Thomas M. Mitchell. 1997. *Machine Learning* (1 ed.). McGraw-Hill, Inc., New York, NY, USA.

[40] Patrick M. Murphy and Michael J. Pazzani. 1991. ID2-of-3: Constructive induction of M-of-N concepts for discriminators in decision trees. In *Machine Learning Proceedings 1991*. Elsevier, 183–187.

[41] Ali Bou Nassif, Ismail Shahin, Imtinan Basem Attili, Mohammad Azzeh, and Khaled Shaalan. 2019. Speech recognition using deep neural networks: A systematic review. *IEEE Access* 7 (2019), 19143–19165. https://doi.org/10.1109/ACCESS.2019.2896880

[42] Daniel W. Otter, Julian R. Medina, and Jugal K. Kalita. 2021. A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems* 32, 2 (2021), 604–624. https://doi.org/10.1109/TNNLS.2020.2979670

[43] J. Ross Quinlan. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, USA. https://dl.acm.org/doi/10.5555/152181

[44] Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence* 1, 5 (2019), 206–215. https://doi.org/10.1038/s42256-019-0048-x

[45] Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. 2022. Interpretable machine learning: Fundamental principles and 10 grand challenges. *Statistics Surveys* 16 (2022), 1–85. https://doi.org/10.1214/21-SS133

[46] D. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning representations by back-propagating errors. *Nature* 323 (1986), 533–536. https://doi.org/10.1038/323533a0

[47] Gabriele Tolomei, Fabrizio Silvestri, Andrew Haines, and Mounia Lalmas. 2017. Interpretable predictions of tree-based ensembles via actionable feature tweaking. In *23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'17)*. ACM Press, Halifax, NS, Canada, 465–474. http://dl.acm.org/citation.cfm?id=3098039

[48] Bhekisipho Twala. 2010. Multiple classifier application to credit risk assessment. *Expert Systems with Applications* 37, 4 (2010), 3326–3336.

[49] Johan Van Benthem and Kees Doets. 2001. Higher-order logic. In *Handbook of Philosophical Logic*, D. M. Gabbay and F. Guenthner (Eds.). Springer Netherlands, Dordrecht, 189–243. https://doi.org/10.1007/978-94-015-9833-0_3

[50] Tim van Gelder. 1990. Why distributed representation is inherently non-symbolic. In *Konnektionismus in Artificial Intelligence und Kognitionsforschung (Informatik-Fachberichte, Vol. 252)*, Georg Dorffner (Ed.). Springer, Salzburg, Österreich, Austria, 58–66. https://doi.org/10.1007/978-3-642-76070-9_6

[51] F. Van Veen and S. Leijnen. 2019. The Neural Network Zoo. Retrieved September 17, 2021 from https://www.asimovinstitute.org/neural-network-zoo

[52] Paul Voigt and Axel von dem Bussche. 2017. *The EU General Data Protection Regulation (GDPR). A Practical Guide*. Springer. https://doi.org/10.1007/978-3-319-57959-7

[53] Laura von Rueden, Sebastian Mayer, Katharina Beckh, Bogdan Georgiev, Sven Giesselbach, Raoul Heese, Birgit Kirsch, Michal Walczak, Julius Pfrommer, Annika Pick, Rajkumar Ramamurthy, Jochen Garcke, Christian Bauckhage, and Jannis Schuecker. 2021. Informed machine learning – a taxonomy and survey of integrating prior knowledge into learning systems. *IEEE Transactions on Knowledge and Data Engineering* 35, 1 (2021), 614–633. https://doi.org/10.1109/TKDE.2021.3079836

[54] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. 2017. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering* 29, 12 (2017), 2724–2743. https://doi.org/10.1109/TKDE.2017.2754499

[55] David H. Wolpert and William G. Macready. 1997. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* 1, 1 (1997), 67–82. https://doi.org/10.1109/4235.585893

[56] Yaqi Xie, Ziwei Xu, Kuldeep S. Meel, Mohan S. Kankanhalli, and Harold Soh. 2019. Embedding symbolic knowledge into deep networks. In *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). Curran Associates, Inc., Vancouver, BC, Canada, 4235–4245. https://proceedings.neurips.cc/paper/2019/hash/7b66b4fd401a271a1c7224027ce111bc-Abstract.html

[57] Cheng Yang, Jiawei Liu, and Chuan Shi. 2021. Extract the knowledge of graph neural networks and go beyond it: An effective knowledge distillation framework. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19–23, 2021*, Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia (Eds.). ACM / IW3C2, 1227–1237. https://doi.org/10.1145/3442381.3450068

[58] Ying Yang, Geoffrey I. Webb, and Xindong Wu. 2010. Discretization methods. In *Data Mining and Knowledge Discovery Handbook, 2nd ed.*, Oded Maimon and Lior Rokach (Eds.). Springer. https://doi.org/10.1007/978-0-387-09823-4_6

[59] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. 2019. Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems* 30, 11 (2019), 3212–3232. https://doi.org/10.1109/TNNLS.2018.2876865

[60] Jan Ruben Zilke, Eneldo Loza Mencía, and Frederik Janssen. 2016. DeepRED – rule extraction from deep neural networks. In *Discovery Science (LNCS, Vol. 9956)*. Springer, Bari, Italy, 457–473. https://doi.org/10.1007/978-3-319-46307-0_29

Received 8 August 2022; revised 11 December 2023; accepted 31 January 2024