



Symbolic knowledge injection meets intelligent agents: QoS metrics and experiments

Andrea Agiollo¹ · Andrea Rafanelli^{2,3} · Matteo Magnini¹ · Giovanni Ciatto¹ · Andrea Omicini¹

Accepted: 17 May 2023
© The Author(s) 2023

Abstract

Bridging intelligent symbolic agents and sub-symbolic predictors is a long-standing research goal in AI. Among the recent integration efforts, symbolic knowledge injection (SKI) proposes algorithms aimed at steering sub-symbolic predictors' learning towards compliance w.r.t. pre-existing symbolic knowledge bases. However, state-of-the-art contributions about SKI mostly tackle injection from a foundational perspective, often focussing solely on improving the predictive performance of the sub-symbolic predictors undergoing injection. Technical contributions, in turn, are tailored on individual methods/experiments and therefore poorly interoperable with agent technologies as well as among each others. Intelligent agents may exploit SKI to serve many purposes other than predictive performance alone—provided that, of course, adequate technological support exists: for instance, SKI may allow agents to tune computational, energetic, or data requirements of sub-symbolic predictors. Given that different algorithms may exist to serve all those many purposes, some criteria for *algorithm selection* as well as a suitable *technology* should be available to let agents dynamically select and exploit the most suitable algorithm for the problem at hand. Along this line, in this work we design a set of *quality-of-service* (QoS) *metrics* for SKI, and a *general-purpose software API* to enable their application to various SKI algorithms—namely, platform for symbolic knowledge injection (PSyKI). We provide an abstract formulation of four QoS metrics for SKI, and describe the design of PSyKI according to a software engineering perspective. Then we discuss how our QoS metrics are supported by PSyKI. Finally, we demonstrate the effectiveness of both our QoS metrics and PSyKI via a number of experiments, where SKI is both applied and assessed via our proposed API. Our empirical analysis demonstrates both the soundness of our proposed metrics and the versatility of PSyKI as the first software tool supporting the application, interchange, and numerical assessment of SKI techniques. To the best of our knowledge, our proposals represent the first attempt to introduce QoS metrics for SKI, and the software tools enabling their *practical* exploitation for both human and computational agents. In particular, our contributions could be exploited to automate and/or compare the manifold SKI algorithms from the state of the art. Hence moving a concrete step forward the engineering of efficient, robust, and trustworthy software applications that integrate symbolic agents and sub-symbolic predictors.

Andrea Agiollo, Andrea Rafanelli, Matteo Magnini: Co-first authors.

Extended author information available on the last page of the article

Keywords Symbolic knowledge injection · Neuro-symbolic integration · Intelligent agents · Quality of service · Efficiency · Robustness · Trustworthiness · PSyKI

1 Introduction

The recent success of Machine and Deep Learning (ML and DL) is endowing computational agents with more and more smart behaviours ranging from text [1] to speech [2] recognition, stepping through image recognition [3], and many more [4]. Typically, such smart behaviours are learnt from data in a semi-automatic way via human-designed malleable predictors—such as neural networks—which can be algorithmically trained to fit that data. This represents a paradigm shift w.r.t. the well-established practice of software engineering where the source code governing computational agents' smart behaviours is designed and produced by human beings. Arguably, the increased interest in ML and DL solutions may be attributed to the groundbreaking performance gain that *data-driven* approaches can bring in comparison to otherwise hard-to-formalise, manually-defined approaches.

However, despite being both flexible and performant, current data-driven solutions come with a number of issues. First, they are *data eager*—meaning that the learning agent should have access to huge amounts of examples concerning the phenomena to learn. When examples are too few, learning cannot happen. Second, learning *takes time*. Unlike humans, learning and exploitation of learnt information are quite distinct stages for computational agents. The trainable components of computational agents are commonly pre-trained by human designers, up to a given performance score, and then provided to the agents for exploitation. Hence, no further learning typically occurs after that. Finally, the benefits of data-driven solutions come at the price of a *reduced understandability* of smart behaviours in the eyes of human users. Modern ML solutions rely upon *sub-symbolic* predictors which work as black boxes—so, humans cannot observe them and tell what a predictor has learnt and how it computes its predictions. This may be troublesome when predictions are used by agents to automate decision-making in critical domains, such as e-health [5] and smart transportation systems [6].

The exploitation of sub-symbolic predictors comes with further issues in the multi-agent systems (MAS) context. As shown by recent surveys [7], it is quite common for the MAS community to represent knowledge and agents' behaviours *symbolically* both at the conceptual and technological level—often relying upon computational logic [8]. This makes the integration of *sub-symbolic* predictors both a conceptual and a technical issue, to be addressed since the earliest design phase down to implementation in any MAS engineering process. As a result, it becomes essential for agent-oriented programming frameworks to be *interoperable* with available ML libraries, and for those libraries to provide a clear and stable API supporting the *automation of ML workflows*—so that intelligent agents can *autonomously* exploit them.

Recently, *symbolic knowledge injection* (SKI) [9–11] has emerged as a possible solution to all the aforementioned issues. SKI is the task of letting sub-symbolic predictors acquire symbolic information and behave *consistently* w.r.t. it. For instance, this may involve a neural network taking into account the information represented by a logic theory when drawing predictions. There, “symbolic” refers to the way knowledge is represented: we consider as symbolic any intelligible language that is naturally interpretable by both human and computational agents. This includes a number of logic formalisms, and excludes the

fixed-sized tensors of numbers commonly exploited in sub-symbolic predictors. Therefore, SKI mechanisms aim at training sub-symbolic predictors towards desirable behaviours.

Thanks to SKI, agents (and their designers) are back in control. The benefits of SKI to the training of ML predictors [12] include the following ones: (1) it mitigates the issues arising from the lack of sufficient amounts of training data—as under-represented situations can be suitably represented in symbols—; (2) it reduces learning time by providing straight away the very knowledge that predictors would otherwise struggle to learn by processing huge amounts of data; (3) it improves predictors' predictive performance in corner cases—as in the case of unbalanced and overlapping classes—; (4) it prevents predictors from working as full black boxes during their training—hence overriding the need for explanations. Furthermore, (5) it harmonises the symbolic and sub-symbolic components of intelligent agents. Hence, agent designers may take advantage of SKI to endow agents with common sense—encoded in some suitable symbolic formalism—, whereas agents themselves may exploit SKI to finely govern their sub-symbolic components—e.g., by tuning them according to their beliefs or desires.

When tuning their sub-symbolic components via SKI, agents will typically aim at maximising predictive performance. It is a common practice to assess SKI mechanisms in terms of the performance gain they introduce w.r.t. some injection-free counterpart [9, 10]. However, performance gain is not the only relevant metrics an agent may intend to optimise. For instance, agents situated into resource-constrained environments may need to minimise the energy required to train/exploit ML predictors, as well as the computational resources required for their execution. Analogously, agents in need to interact with human beings may be aiming at maximising the intelligibility of their decision-making processes. Overall, there are several aspects of sub-symbolic predictors that agents could optimise via SKI. Along this line, as part of our recent research activities [13], we sketched a set of quality-of-service (QoS) metrics for SKI covering several aspects—ranging from energy-related to computational-cost-related ones via comprehensibility-related ones. Unfortunately, at that time we could not assess QoS metrics empirically, due to the lack of general-purpose software technologies supporting them.

The lack of viable software technologies for SKI is preventing not just the assessment of QoS metrics from [13], but also—and in the foremost place—the effective exploitation of SKI methods in MAS. However, as further discussed in [11], SKI methods from the literature share a general workflow, which can be briefly summarised as follows: (1) identify a suitable predictor w.r.t. the learning task at hand; (2) attain some symbolic knowledge aimed at describing relevant situations; (3) apply some SKI method to the given predictor and knowledge, hence generating a new predictor that encapsulates the knowledge; (4) train the new predictor on the available data, as usual. Notably, the last two steps may be cyclically repeated by an agent until some target QoS score is reached. Hence, in principle, SKI methods are interchangeable at the functional level as well as at the assessment level. Along this line, we designed a unified open source software library for SKI—namely, PSyKI¹—supporting the interchange, comparison, and exploitation of SKI methods in arbitrary ML workflows [11]. However, support for QoS-based assessment is currently missing.

Accordingly, in this paper we extend our previous work by proposing a full modelling of the QoS metrics for SKI, as well as their empirical evaluation² via PSyKI. To serve this purpose, we also extend PSyKI design, API, and codebase to support our QoS metrics.

¹ The code is public available at github.com/psykei/psyki-python.

² Experiments are available at github.com/pikalab-unibo/ski-qos-jaamas-experiments-2022.

Our empirical analysis demonstrates both the soundness of the proposed metrics and the versatility of PSyKI as the first software tool supporting the application, interchange, and numerical assessment of SKI techniques. Our proposals are—to the best of our knowledge—the first attempt to introduce QoS metrics for SKI, along with the software tools enabling their *practical* exploitation by both human and computational agents.

The paper is organised as follows. Section 2 introduces some relevant definitions and summarises the background on SKI methods. Section 3 formally defines the QoS metrics, whereas Sect. 4 overviews PSyKI and describes the integration of QoS metrics. Section 5 outlines the experiments and their design, and discusses the results. Finally, Sect. 6 summarises the key findings and contributions, by highlighting the importance of the new QoS metrics in effectively evaluating the strength of SKI mechanisms.

2 Background and definitions

The benefits of sub-symbolic predictors in MAS come along with the issues deriving from their black-box nature and uncertain optimisation processes. This is why incorporating symbolic knowledge into the sub-symbolic prediction process could bring about a number of advantages. For instance, predictors may be able to make informed decisions based on prior knowledge, reducing the chances of producing unexpected results. Moreover, the injection of symbolic knowledge often results in improved prediction performance, as the predictors are better equipped to handle data with inherent structure and meaning. Therefore, a number of recent works [9, 10] have leveraged *symbolic knowledge injection* to mitigate the common problems of sub-symbolic predictors (lack of interpretability, poor generalisation, fuzzy optimisation procedure, etc.). The underlying idea is to enable the sub-symbolic predictor to take into account some prior symbolic knowledge when drawing its predictions, thus making the predictor more controllable.

The practice of SKI involves a rather simple workflow, yet it may rely on several different injection algorithms, often tailored on specific sorts of predictors or symbolic languages. Differences among those algorithms can be relevant, especially w.r.t. to *how* they perform injection. Hence, we can broadly define SKI as “any algorithmic procedure affecting how sub-symbolic predictors draw their inferences so that predictions are either computed as a function of, or made consistent with, some given symbolic knowledge”.

More formally, given an injection procedure \mathcal{I} , some symbolic knowledge K , and a sub-symbolic predictor N aimed at solving some supervised learning task, we define the “knowledge-aware” predictor \hat{N} as the result of the application of \mathcal{I} to K and N :

$$\hat{N} = \mathcal{I}(K, N)$$

There, we call N the *uneducated* predictor—as it has not yet undergone injection—, and \hat{N} the *educated* one.

Focussing on the inputs of SKI—namely, the symbolic knowledge K and the sub-symbolic predictor N —, nearly all SKI methods and techniques available in the literature assume that: (1) K is a *logic* knowledge base (KB, henceforth) of logic formulæ, encoded via some subset of first-order logic (FOL) [14], while (2) N is a neural network (NN). To support this statement, Table 1 reports a sample of the most relevant SKI techniques from the literature, pointing out the sorts of knowledge and predictor they support.

Should we speculate on the possible motivations behind the choice of FOL and NN, we would argue that logic brings great flexibility in representing knowledge in a way which is

Table 1 Summary of relevant SKI techniques (not exhaustive)

Algorithm	Year	Typology	Predictor	Knowledge	Task
KBANN [17]	1990	S	NN	P	C
Gaussian NN [18]	1992	C + S	NN	P	C + R
C-IL ² P [19]	1999	S	NN	P	C
Fibred NN [20, 21]	2004–2005	S	NN	FOL	AR
CODL [22]	2007	C	MNN	P	C
RESCAL [23]	2011	E	NN	KG	LP
TRESCAL [24]	2014	E	NN	KG	LP
KALE [25]	2016	C	NN	KG	C
Student-Teacher [26, 27]	2016	S	NN	FOL	C
HoE [28]	2016	E + C	NN	KG	LP
SBR (a) [29]	2017a	C	KM	FOL	C
SBR (b) [9]	2017b	C	NN	P	C
SLF [10]	2018	C	NN	P	C
δ ILP [30]	2018	C + S	NN	D	C
LRNN [31]	2018	S	NN	D	C
RotatE [32]	2019	E + C	NN	KG	LP
LTN [33]	2020	S	NN	FOL	C + R
KINS [34]	2022	S	NN	P	C + R
KILL [35]	2022	C	NN	P	C

In column *Typology* S stands for structural, C for constraining, and E for embedding. In column *Predictor* NN stands for (deep) neural network, MNN for Markov NN, and KM for kernel machine. In column *Knowledge* FOL is first order logic, P is propositional logic, KG is knowledge graph, D is Datalog. In column *Task* C stands for classification, R for regression, LP for link prediction, and AR for automated reasoning

similar to how humans reason, whereas neural networks bring about malleability, composability, and trainability to intelligent systems—as they can be structured in various ways to serve diverse purposes.

Many algorithms may fit the definitions above—mostly differing for the particular sort of logic formalism, injection strategy, or neural network they support. For a more detailed discussion on SKI algorithms see [12, 15, 16].

2.1 Knowledge injection workflow

SKI assumes the input knowledge consists of crisp logic formulæ expressed in a logical language of choice. Such formulæ must somehow be converted into numeric form for injection to take place. Later on, injection is performed by training neural networks as usual. In other words, SKI is a process affecting networks *before* and *during* training.

Overall, SKI relies on three basic operations, namely *parsing* (Π), *fuzzification* (ζ), and *embedding* (E).

The first step of any SKI method is *parsing* the input formulæ, hence producing a machine-interpretable and -browsable representation—namely, *abstract syntax trees* (AST). AST are then visited to produce a *numeric* representation of the input formulæ

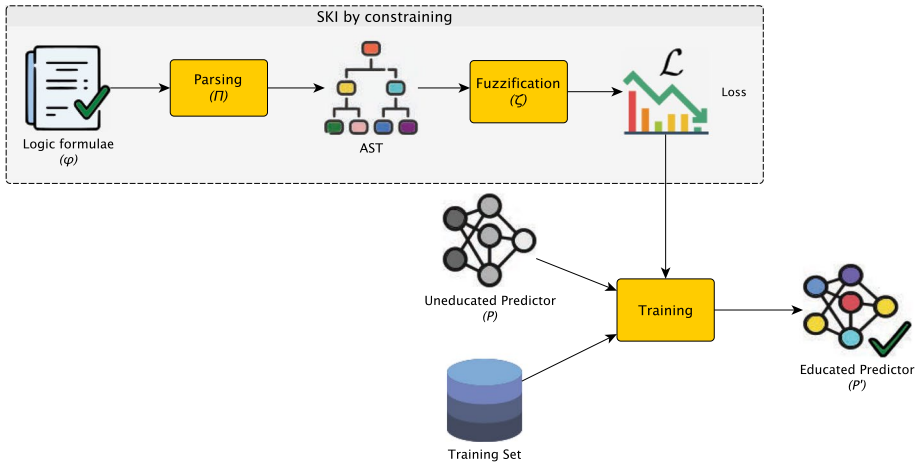


Fig. 1 Symbolic knowledge injection via constraining: data flow

either consisting of functions of real numbers (e.g. a loss function, a neural network structure), or an array of real numbers.

Fuzzification is the process of converting some AST formula into a function of the form $f : \mathbb{R}^n \rightarrow \mathbb{R}$, whose input values are numeric interpretations of the original formula, whereas their outputs are either truth degrees (cf. [35])—e.g. 1 means true, 0 means false, $x \in [0, 1]$ means “true with $x\%$ probability”—, or penalties (cf. [34])—e.g. 0 means “no penalty”, $x \neq 0$ means “penalty proportional to $|x|$ ”. This is required when converting formulae into loss functions (like in constraining methods) or activation functions (like in structuring methods).

Embedding is the process of converting some formula’s AST into a numeric array of fixed size. This is necessary when converting formulae into numeric datasets for training.

As also highlighted in Table 1, there are three sorts of SKI methods: those that perform injection *constraining* during the training of neural networks, those that affect their internal *structure*, and finally those that perform *embedding*.

2.2 Categorisation of injection methods from the literature

In the remainder of this section we delve into the details of the various injection strategies exploited by SKI, and elaborate on how the overall performance of SKI techniques can be assessed.

2.2.1 Constraining neural networks

The key idea behind SKI techniques of this sort is to *steer* the learning process of a neural network to make it behave consistently w.r.t. some given logic formulae. This is achieved by penalising the network during training, whenever it violates the logic formulae. Figure 1 provides an overview of the approach.

A common way to penalise the network under training is by altering the loss function [9, 10, 36]. The neural network training process essentially consists in the use of gradient descent [37], i.e. an optimisation process where the weight of NN synapses are iteratively

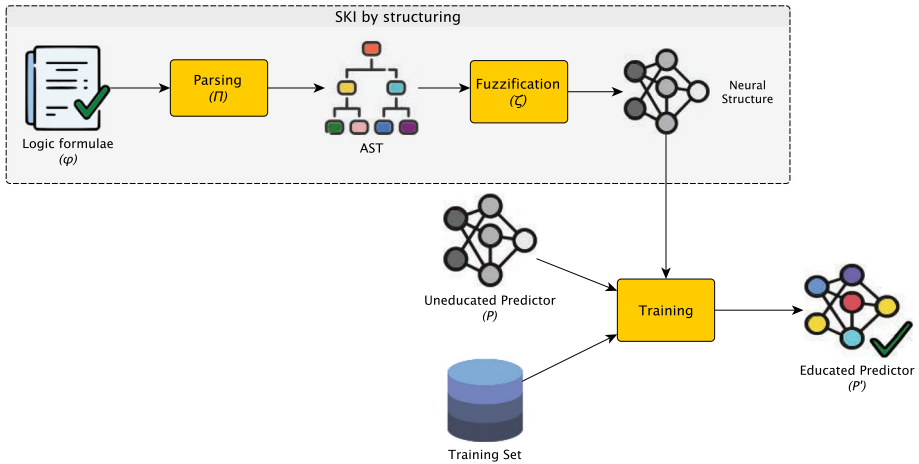


Fig. 2 Symbolic knowledge injection via structuring: data flow

modified so as to minimise a loss function. Most commonly, the loss function quantifies the overall predictive error of the network: the greater the error, the greater the loss. However, when SKI is applied, the loss function also takes into account the consistency of the logic formulae. In this way, the learning process not only minimises the network error w.r.t. data, but also its error w.r.t. symbolic knowledge. In other words, the predictor is *constrained* to be compliant with the prior knowledge up to a certain extent.

The underlying assumption behind injection mechanisms of this kind is that logic formulae should be converted into functions of real numbers of the form:

$$f : \mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$$

where \mathcal{X} is the same input space of the network, and $\mathbb{R}_{\geq 0}$ is the set of non-negative numbers—here representing penalties. There, for any given input vector \mathbf{x} , the value $f(\mathbf{x})$ represents the discrepancy among the network prediction corresponding to \mathbf{x} and what the logic formulae prescribe for \mathbf{x} . Hence, $f(\mathbf{x}) = 0$ means that the network is behaving consistently w.r.t. the formulae, hence it should get no penalty. Conversely, $f(\mathbf{x}) > 0$ means that the network behaviour is deviating the formulae, hence it should be penalised.

Some relevant SKI algorithms based on constraining can be found in [9, 10, 25, 29, 30, 38].

2.2.2 Structuring neural networks

The key idea behind SKI techniques of this sort is to *construct* (a portion of) the neural network undergoing injection in such a way to make it reflect some given logic formulae [18, 20, 30, 39, 40]. The resulting network is then trained as usual. However, given that (part of) its internals are tailored on the logic formulae, the network is expected to have higher predictive performance—or at least require less training efforts to reach good performance scores—in all situations which are described by the logic formulae. Figure 2 provides an overview of the approach.

The underlying assumption behind structuring SKI methods is that (a portion of) a neural network can be constructed to *mimic* the evaluation of one or more logic

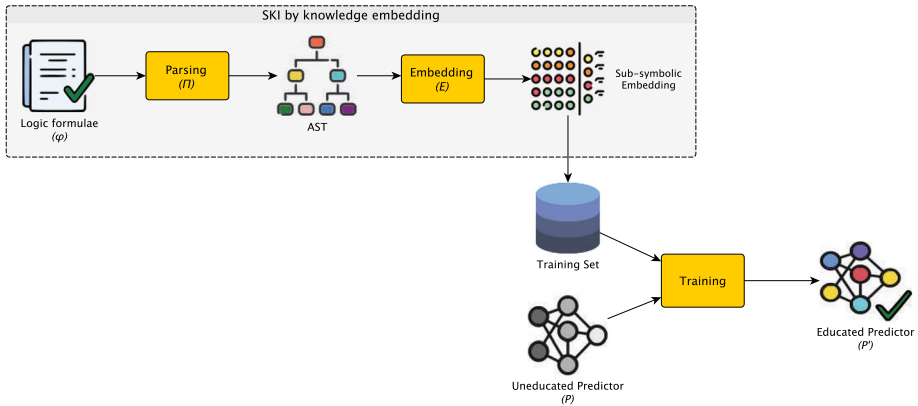


Fig. 3 Symbolic knowledge injection via embedding: data flow

formulae. This is commonly achieved by letting neurons and synapses represent either logic variables or combinations of logic expressions via logic connectives or arithmetic operators. Methods may then decide to keep the weights of the structured portion of network free to vary during training, in order to let them adapt to the peculiarities of the training data at hand.

Some relevant SKI algorithms based on structuring can be found in [17–19, 21, 26, 27, 31, 33].

2.2.3 Embedding knowledge into neural networks

The key idea behind SKI techniques of this sort is to convert symbolic knowledge into numeric-array form to be used as training data [41–43]. Predictors trained with such techniques are usually used as logic reasoning engines. Figure 3 provides an overview of the approach.

The underlying assumption behind embedding SKI methods is that input knowledge can be represented as a (possibly multi-dimensional) array of numbers. This, in turn, requires the knowledge to *ground* (i.e. variable free)—a requirement which heavily limits what logic can actually represent. So, in practice, embedding techniques are commonly applied to simple (i.e. less expressive) logics such as description logics. There, symbolic information consists of *knowledge graphs* [44], where nodes represent entities and edges represent relations among those entities. The graphs' adjacency matrices are essentially numeric arrays—and this is one of the tricks exploited by embedding-based SKI methods.

Some relevant SKI algorithms based on knowledge graph embedding can be found in [23, 24, 28, 32].

2.3 Injection assessment

It is common for works in the SKI realm to measure the strength of their mechanism as the gain in performance achieved by the SKI predictor against its *uneducated* counterpart. In that case, the effectiveness of the injection mechanism \mathcal{I} when applied to a neural

network N to inject the knowledge K is measured via some performance score π (accuracy, F1-score, MSE, etc.), aimed at assessing the performance of N with respect some test dataset T . More formally:

$$\epsilon_{K,N,\pi,T}(\mathcal{S}) = \pi(\mathcal{S}(K, N), T) - \pi(N, T) \quad (1)$$

In other words, the effectiveness of some injection mechanism \mathcal{S} may be assessed differently depending on which knowledge base, neural network, and dataset it is applied to.

While being indicative of the quality of the SKI approach w.r.t. predictive performance, that metric does not capture every aspect of the knowledge injection, as there exist multiple properties that one may be willing to optimise through SKI—see Sect. 3.1. Due to the sudden rise in research interest towards sustainable AI approaches [45], there exists the opportunity to analyse if and how SKI brings about benefit in terms of computations, energy consumption, and data required to train and deploy sub-symbolic approaches. In the remainder of this paper we identify other metrics to reliably measure the performance of SKI.

3 SKI quality-of-service metrics definition

In this section we propose and analyse the novel set of metrics for identifying the quality of SKI systems. An overview of our proposals, along with a brief classification, is provided in Sect. 3.1. Roughly speaking, we introduce metrics for measuring SKI method's *efficiency*—under multiple goodness criteria.

3.1 Overview

The current practice of SKI assessment relies exclusively on measuring improvements in the predictive performance of some educated predictor over an equivalent uneducated counterpart. However, predictive performance is not the only relevant benefit of SKI one may be willing to measure.

There exist multiple aspects of neural predictors which may be affected by SKI—and for which metrics should be defined. Just to name a few, SKI may affect the memory footprint, the latency, as well as the data and energy requirements of the predictors it is applied to. Overall, all such properties contribute to what we informally call a predictor's *efficiency*. In the remainder of this paper we rely on the following *efficiency properties*:

- *memory footprint*, i.e., the size of the predictor under examination;
- *latency*, i.e., the time required to run a predictor for inference;
- *data efficiency*, i.e., the amount of data required to train the predictor;
- *energy consumption*, i.e., the amount of energy required to train/run the predictor;

other than, of course:

- *predictive performance*, e.g. accuracy, F1-score, mean absolute/squared error, etc.

For the sake of brevity, we also denote as *efficiency metrics* any function aimed at measuring some efficiency property.

Efficiency metrics provide a score measuring how much some efficiency property P of a given uneducated predictor N improves in its educated counterpart \hat{N} . Of course, the resulting score may be largely influenced by a number of different aspects, such as:

- (A1) *Knowledge quality and coverage.* The educated predictor \hat{N} is attained by injecting some input knowledge K . Furthermore, both N and \hat{N} are aimed at addressing the same learning task—say, classification or regression—, and they are both trained upon some training dataset D , which describes the task. Questions that may arise are, for instance: (1) are K and D coherent?, (2) is K covering situation which the data in D exemplifies?, (3) is K consistent, coherent, and correct? (4) can we say the same for D ? Regardless of the particular efficiency property P being measured, the resulting score may greatly vary depending on the answers to these questions. So, in other words, efficiency measures depend on the particular input knowledge (K) and data (D) being used during SKI.
- (A2) *Baseline quality.* As both the educated (\hat{N}) and uneducated (N) predictors are targeting the same learning task, one may wonder if N is adequate enough to address that learning task. In this setting, questions that may arise are: (1) is N biased [46] in statistical sense? (2) in case it is, can we expect \hat{N} to carry any observable improvement on some efficiency measure P ? (3) can we expect \hat{N} to carry any observable improvement on some efficiency measure P ? (4) even in case where N is not biased, is the selected injection mechanism \mathcal{I} adequate for N ? From these questions we understand that efficiency measures may also depend on the nature of the input predictor (N), and, of course, on the injection mechanism of choice (\mathcal{I}).
- (A3) *Task at hand.* The learning task targeted by both N and \hat{N} determines the training dataset as well as the test dataset T . The choice of T impacts the assessment of both N and \hat{N} . Therefore, it may impact the score of any efficiency measure as well. So, efficiency measures may finally also depend on the target learning task, and, consequently on the test data (T).

Summarising, efficiency measures assess some injection method \mathcal{I} in a very specific setting that depends on (1) the particular knowledge to be injected, (2) the sort of predictor undergoing injection, (3) the training and (4) test data adopted for training. In other terms, any efficiency measure should be parametric w.r.t. K , N , D , and T .

Accordingly, in the following we propose the implementation and formalisation of metrics to assess the efficiency of SKI. In particular, as discussed at the beginning of this section, *memory footprint*, *latency*, *energy consumption*, and *data efficiency* are introduced as key performance indicators of SKI. Our objective is to assess the efficiency of SKI in terms of computational resource usage, and to provide insight into how these metrics could be used to inform the design and optimisation of SKI-based systems, with a particular emphasis on their potential benefit within MAS frameworks. Notably, we believe that an in-depth understanding of the trade-off between performance and efficiency is essential for the implementation of AI predictors in MAS frameworks.

3.2 Memory footprint

In the context of MAS, sub-symbolic predictors are gaining importance as the field moves towards more efficient and sustainable AI. As the demand for AI predictors that

can operate on resource-constrained devices—such as IoT devices, edge devices, and mobile devices—continues to rise, researchers have focused on developing solutions that require less memory and computational resources [47–49].

To address those concerns, several metrics for measuring the memory footprint of AI predictors—in particularly the sub-symbolic ones – have been recently proposed in the literature [50–52]. For instance, in [13], the authors propose measuring neural networks' memory footprint by counting the amount of parameters they are composed by—i.e., essentially, the amount of synapses composing each neural network. Alternatively, some authors leverage metrics such as Floating Point Operations (FLOPs) [53] or Multiplication Addition Computations (MACs) [54], which measure the amount of total operations or multiplications and additions required to perform a single inference respectively. MACs consider solely multiplications and summations as they represent the most common computations in NNs. These measures are indicative of the amount of memory required either to fit the whole sub-symbolic predictor—total number of parameters—or to run it—FLOPs and MACs. Even though they are intuitive, those metrics are also effective for measuring predictors complexity and the overall computational memory efficiency. Here, we consider leveraging on such measures to analyse the efficiency gain of SKI approaches. In other terms, we consider the ability of SKI mechanisms to produce *lightweight* sub-symbolic predictors—in terms of memory occupation.

The key insight here is that knowledge injection may lift part of the learning burden from the predictor at hand, by relieving the network from the need to learn complex or data-uncovered notions via trial-and-error. Indeed, the a-priori concepts carried by the input knowledge might now be injected instead of being learnt in a data-driven way. As a result, the amount of notions that sub-symbolic predictors must learn in a data-driven way might be significantly reduced. Fewer concepts to be learned are typically associated with fewer parameters, FLOPs, and MACs—or, in other words, a smaller memory footprint [55]. In the context of SKI, we define the memory footprint *improvement* score $\mu_{\Psi,K,N}(\mathcal{S})$ as the amount of memory saved by the educated network \hat{N} w.r.t. its uneducated counterpart N . The higher the score, the more memory efficient the educated predictor is w.r.t. the uneducated one. However, as one may measure the memory footprint of a sub-symbolic predictor in different ways – e.g., by counting the number of parameters, FLOPs or MACs—, our scoring function is parametric in Ψ —which represents the memory footprint metric of choice. More formally:

$$\mu_{\Psi,K,N}(\mathcal{S}) = \Psi(N) - \Psi(\mathcal{A}(K, N)) \quad (2)$$

where $\hat{N} = \mathcal{A}(K, N)$ represents the educated predictor attained by injecting K into N .

It is worth noticing how the proposed memory footprint score may be influenced by quality and coverage of the input knowledge (A1), as well as by the memory footprint of the input predictor N (A2). About (A1), the reason is simple: the better the input knowledge, the lower the expected memory requirements of the educated predictor. Similarly, as far as (A2) is concerned, the better the input predictor, the lower the expected memory footprint improvements of the educated predictor. However, one may also notice from Equation (2) that our memory footprint score is not parametric when the current task is taken into account (A2). The reason is simple: the memory footprint of a neural network is *not* task-dependent, as it is a structural property of the neural network itself.

Finally, we stress that memory footprint of the educated predictor is expected to be lower than the one of the uneducated predictor. Indeed, our score is measuring the memory footprint *improvement*. A negative score $\mu_{\Psi,K,N}(\mathcal{S})$ means that the educated predictor is

more memory hungry than the uneducated one—i.e., that the SKI approach is not effective in reducing the memory footprint of the input predictor.

3.3 Energy consumption

The relationship between MAS and energy consumption is complex and articulated. In fact, to function effectively in resource-constrained environments, MAS require AI systems that consume a reduced amount of energy. Moreover, the distributed nature of MAS and the increasing demand for AI across a variety of applications drive the need for scalable and power-efficient solutions. This need is, however, aggravated by the dynamic and real-time requirements of many MAS applications, which can result in high energy consumption due to computational and resource requirements. The complexity of a MAS, with its multiple agents and agent interactions, adds a new level of difficulty, especially when processing large amounts of data or executing complex algorithms. Altogether, those factors make energy consumption a crucial aspect of the design and implementation of AI systems in MAS frameworks.

Overall, there are numerous ways to address this problem. One approach could be to use energy-efficient hardware, such as low-power processors, or to use distributed and federated learning techniques that can distribute computation complexity across multiple devices [56]. Another approach could exploit more efficient algorithms and data structures so as to reduce the amount of computation required by agents to process data. Along that line, the integration of sub-symbolic predictors, which require less memory and computational resources than conventional symbolic AI approaches, could meaningfully reduce energy consumption. However, improvements can still be made from an energy point of view by making the sub-symbolic systems more efficient. For instance, several techniques rely on ad-hoc strategies to compress or optimise sub-symbolic predictors.

In this context, we see SKI approaches as providing MAS designers with a huge opportunity. In fact, the introduction of injection mechanisms in the data-driven pipeline of sub-symbolic training mechanisms may reduce the amount of computations required to train and run sub-symbolic predictors. Indeed, knowledge injection reduces the complexity of the learning process, providing another source of knowledge other than the training data itself. Thus, one may be interested in assessing whether and to what extent SKI mechanisms contribute to reducing the amount of computations required by a sub-symbolic predictor along its life-cycle.

We propose a new score aimed at measuring the energy consumption of SKI approaches. This is tightly related with memory footprint score from Sect. 3.2. Indeed, it is usually the case for smaller predictors to require fewer amounts of energy to train and run. However, there may exist memory efficient predictors requiring a higher amount of energy to train and run, such as sparse ones. Indeed, sparsity induces a lower amount of operations, but is not usually effectively implemented at hardware level, increasing power consumption [53]. Therefore, energy consumption is a property which is worth to be measured by itself.

In order to analyse energy consumption as well as the possible improvements that SKI could bring about, we first need to define the life cycle of AI predictors, analysing hungriness of each component resource. In order to build and deploy a data-driven AI solution, a number of stages need to be completed, namely:

1. *Model definition*, where data scientists analyse the task at hand and select the most adequate sorts of sub-symbolic predictors, and the most promising hyperparameters assignments for those predictors.
2. *Model training*, where the parameters of the sub-symbolic predictor of choice are automatically tuned on the training data via some sort of training algorithm. There, the amount of training samples (as well as their dimensionality) may impact energy consumption. Indeed, training algorithms commonly require running the predictor on the data and updating it several times.
3. *Model testing*, where the predictor is tested against a—limited—set of testing samples to check if the performance are satisfactory. As for the training case, energy consumption here may be affected by the amount (and dimensionality) of testing samples.
4. *Model deployment*, where the predictor runs multiple times, which a frequency which really depends on the specific application at hand

From the definition of the data-driven AI life-cycle, it is possible to highlight that the *training* and *deployment* phases are the most resource hungry. Indeed, training requires a huge—yet predictable—amount of predictor executions and updates, whereas deployment might be very energy demanding depending on the predictor usage frequency and life expectation—which are typically hard to anticipate.

Accordingly, as far as energy consumption is concerned, we are interested in measuring the energy consumption of the training and deployment phases, individually. More precisely, for the training phase, we are interested in measuring the energy consumption of the training algorithm itself, hence excluding the cost of the inferences drawn during the training process—as their cost is expected to be analogous to the one of the deployment phase.

Notably, this distinction allows us to evaluate the impact of SKI during both the training and deployment phases—which may, in general, be significantly different. In fact, we expect SKI to decrease the energy consumption of the deployed predictors, at the price of an increased energy consumption of the training phase.

Delving into the details of the energy consumption measurements, we start by defining the Υ^i score, aimed at measuring the average energy consumed by a sub-symbolic predictor N on a per-single-inference basis:

$$\Upsilon_v^i(N, T) = \frac{1}{|T|} \sum_{t \in T} v(N, t) \quad (3)$$

Our definition assumes a function $v(N, t)$ is available to measure the energy consumption of a single forward run of a sub-symbolic predictor N on a single sample t . Such a function may for instance estimate the heat dissipated by the hardware running the predictor, during the single inference $N(t)$. Under that assumption, Eq. (3) measures the *average* energy consumption of a sub-symbolic predictor N on a test dataset T composed by several samples.

We now define the Υ^t score, aimed at measuring the average energy consumed while training a sub-symbolic predictor N on a training dataset T :

$$\Upsilon_{v,\gamma}^t(e, N, T) = \frac{\gamma(e, N, T)}{e \cdot |T|} - \Upsilon_v^i(N, T) \quad (4)$$

Our definition assumes the training involves e epochs, and that during each epoch the whole training set T is used to update the predictor N . The definition also assumes $\gamma(e, N, T)$ is a function estimating the overall energy consumed by the training phase as

whole—including the energy consumed by the inferences drawn during the training process. Similarly to v , function γ may for instance estimate the heat dissipated by the hardware running the predictor, during the whole training process. Under such assumptions, Eq. (4) measures the *average* energy consumption required by the predictor N for a single update, during its training on the dataset T .

We can now define the energy consumption *improvement* of a SKI mechanism as the amount of energy saved by the educated predictor, compared to its uneducated counterpart. Again, we distinguish between energy consumption during training and energy consumption during inference. Along this line, we introduce two scores, namely $\epsilon_{v,K,N,T}^i(\mathcal{S})$ (resp. $\epsilon_{v,\gamma,K,N,T}^t(\mathcal{S})$), aimed at measuring the energy consumption improvement of a SKI mechanism \mathcal{S} , during inference (resp. training). More formally:

$$\begin{aligned}\epsilon_{v,K,N,T}^i(\mathcal{S}) &= Y_v^i(N, T) - Y_v^i(\mathcal{S}(K, N), T) \\ \epsilon_{v,\gamma,e,K,N,T}^t(\mathcal{S}) &= Y_{v,\gamma}^t(e, N, T) - Y_{v,\gamma}^t(e, \mathcal{S}(K, N), T)\end{aligned}\quad (5)$$

where $\hat{N} = \mathcal{S}(K, N)$ represents the educated predictor attained by injecting K into N , and T is a reference dataset of choice—most commonly, the training set in the case of ϵ^t , and the test set in the case of ϵ^i .

It is worth noticing how the proposed energy consumption scores may be influenced all aspects (A1)–(A2). About input knowledge (A1) the reason is simple: the more complex the input knowledge, the higher (resp. lower) the expected energy consumption of the educated predictor during training (resp. inference). Similarly, as far as the input predictor is concerned (A2), the more energy-hungry it is, the higher we expect the educated predictor's energy consumption improvements to be. Lastly, the task at hand (A2) has a clear effect on our scores, as they are both parametric in the dataset—energy consumption improvements are typically task-specific.

3.4 Latency

The amount of time required to draw a single prediction is one of the most relevant and impactful efficiency measures for sub-symbolic predictors. In what follows, we refer to such time-lapse as *latency*. A small latency indicates that a sub-symbolic predictor is able to compute relevant predictions in useful time—which is an important property in real-world applications. For instance, low latency is essential in those scenarios where human lives depend on the timely response of some AI system, such as intelligent transportation [6] and e-health [5]. Moreover, latency assumes a relevant role in multi-agent scenarios, where collaboration between multiple intelligent entities is required, and there can not exist lag between them due to lengthy computations [57]. Also, the processing of large amounts of data and execution of complex algorithms, such as those used in decision-making, can result in increased latency as the system struggles to keep up with the demands of the task. As a result, MAS complexity can contribute significantly to increase latency. This is why recent research efforts in the AI field are focussing on time-sensitive predictors.

One possible solution available to address this problem is the use of SKI approaches. By incorporating symbolic representations, SKI approaches can reduce the amount of computation required to process data, leading to reduced latency. Furthermore, the use of symbolic representations could help to simplify the complexity of the system, making it easier to predict the behavior of the system and identify the root causes of an increased latency.

As a result, we believe it is crucial to measure latency in order to assess the computational efficiency of present AI systems.

In the remainder of this section we assume latency to be computed by averaging the time required to draw a number of predictions from a reference test dataset. More formally, we define the latency of a predictor N as the average time required to draw a single prediction from a dataset T :

$$\Lambda(N, T) = \frac{1}{|T|} \sum_{t \in T} \Theta(N, t) \quad (6)$$

where $\Theta(N, t)$ represents the time required to draw a prediction from N on the input t .

As far as SKI is concerned, we are interested in assessing the *latency gain* brought by a given SKI mechanism \mathcal{S} w.r.t. some uneducated predictor. Along this line, one may be interested in figuring out whether injection increases or decreases the latency of a given predictor. Hence, we define the latency gain $\lambda_{K,N,T}(\mathcal{S})$ introduced by some SKI method \mathcal{S} as the average difference between the inference time of the educated predictor \hat{N} and its uneducated counterpart N , over a reference test dataset T . More formally:

$$\lambda_{K,N,T}(\mathcal{S}) = \frac{1}{|T|} \sum_{t \in T} (\Theta(N, t) - \Theta(\hat{N}, t)) = \Lambda(N, T) - \Lambda(\hat{N}, T) \quad (7)$$

where $\hat{N} = \mathcal{S}(K, N)$ represents the educated predictor attained by injecting K into N .

Similarly to the energy measurement, the latency metric is tightly related to the complexity of the educated sub-symbolic predictor and therefore with memory footprint. However, like energy consumption, latency is not always directly proportional to the amount of operations that construct the predictor at hand. Sparsely-structured operations might slow down the inference process due to their inefficient computation at hardware level. Moreover, input data complexity and quality might alter the latency achieved by the predictor. Indeed, inference over different—yet structurally analogous—samples may take vastly different timings, as shown in the attack proposed in [58].

It is worth noticing how the proposed latency score may be influenced all aspects (A1)–(A2). About input knowledge (A1), we argue it may have both a positive and a negative effect on the latency gain. In fact, on the one hand, some SKI mechanisms might introduce additional computations—such as the ones required to process the input knowledge K in structuring methods—see Sect. 2.2.2. We expect this effect to be magnified in the case of large knowledge bases, as the number of operations required to process them is expected to be higher. On the other hand, SKI systems might also reduce the inference time of the given predictor, by reducing the number of computations required to draw a prediction—likely, at the expense of higher training times. As far as the input predictor is concerned (A2), the more time-consuming it is, the higher we expect the educated predictor's latency gain to be. Lastly, the task at hand (A2) has a clear effect on our score, as latencies are computed over task-specific test sets.

3.5 Data efficiency

Data efficiency is a critical aspect of MAS, as the amount of data generated and processed by these systems can be substantial. Inefficient data management can result in increased latency, decreased accuracy, and increased energy consumption, all of which can negatively impact the performance of MAS.

Sub-symbolic predictors which rely on data-driven training algorithms, can provide groundbreaking performance and flexibility to MAS, but the data-driven procedure comes with several shortcomings. These predictors require collecting significant amounts of data samples for each task to be tackled, leading to increased data storage and processing requirements. Furthermore, not only the quantity but also the *quality* of the data—here intended as its representativeness of the task at hand—is crucial for the predictor to learn effectively. All such requirements make the data collection process time-costly—and depending on the application—possibly affected to subjectivity or uncertainty—e.g., emotion recognition [59].

For all these reasons, recent research efforts have focused on proposing data-frugal predictors [60]. Among them, knowledge injection mechanisms play a significant role [10]. Indeed, leveraging a-priori knowledge, SKI relieves the learning process from part of its computational burdens. Concepts that an uneducated predictor would need to learn from data might now be injected into the educated predictor, instead. Hopefully, this would let the educated predictor's learning process require lower amounts of data to attain acceptable performance levels. In this sense, SKI might be considered as a data-efficiency mechanism.

We are here interested in computing the *data-efficiency gain* brought by a given SKI mechanism w.r.t. some uneducated predictor. To do so, we firstly need to define the *data footprint* of a given predictor. Informally, the data footprint of a predictor N is the amount of data it requires to be trained to reach a certain performance level. Hence, assuming that a predictor N is trained on a dataset D —of samples of potentially different dimensions—, via some training process involving e epochs, and that it reached a performance level $\pi(N, T)$ over a test set T —and according to some test dataset T —, we define its data footprint as follows:

$$\Delta_{\pi}(e, N, D, T) = \frac{e}{\pi(N, T)} \sum_{d \in D} \beta(d) \quad (8)$$

where d is a single training sample, and $\beta(d)$ is the amount of bytes required for its in-memory representation, and π is some performance score of choice. As the reader may notice, the data footprint is *directly* proportional to the number of epochs e , to the size of the training set, and to its dimensionality; whereas it is *inversely* proportional to the performance score of the resulting predictor.

We define the data-efficiency gain $\delta_{e,K,N,D,D',T}(\mathcal{S})$ of a given SKI mechanism \mathcal{S} as the difference between the data footprint of the uneducated predictor N —trained upon some dataset D —and that of the educated predictor $\mathcal{S}(K, N)$ —trained upon some other dataset D' . The score assumes that the two predictors have been trained for the same number of epochs e , and that their performance is assessed using the same performance score π , on the same test set T —in order to keep the comparison fair. More formally:

$$\delta_{e,K,N,D,D',T}(\mathcal{S}) = \Delta_{\pi}(e, N, D, T) - \Delta_{\pi}(e, \mathcal{S}(K, N), D', T) \quad (9)$$

The simplest approach to improve data efficiency in SKI mechanisms is to reduce the amount of samples that compose the training dataset—i.e. $|N|$ in Eq. (8). However, one may also consider the option of decreasing the size of samples either by reducing their dimensionality or by compressing their representations—in a nutshell, by reducing $\beta(d)$ for all $d \in D$.

To increase the data-efficiency gain, one may also consider engineering SKI and, consequently, the educated predictor. Along this line, the best strategy consist in reducing the

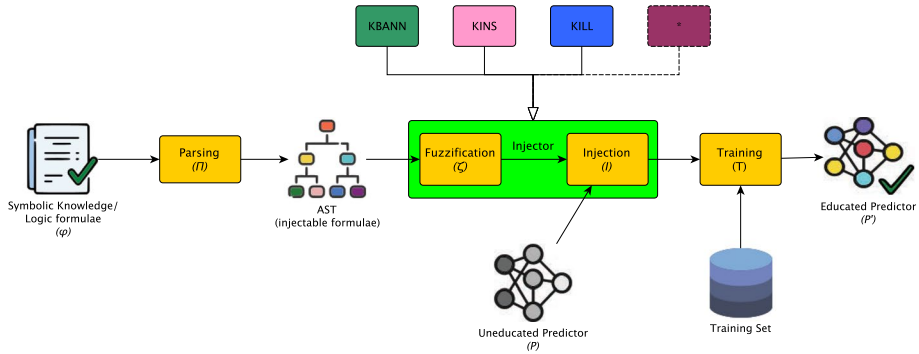


Fig. 4 PSyKI design. Each SKI algorithm is follow the workflow represented in the figure The four yellow boxes represent the four main steps of the workflow. The first step is the parsing (Π) of the symbolic knowledge. The second step is the fuzzification (ζ) of the parsed knowledge. The third step is the injection (\mathcal{I}) of the fuzzified knowledge into the uneducated target predictor (P). The fourth step is the training (\mathcal{T}) of the new predictor, making it educated (P')

size of the training set D' for the educated predictor by letting the input knowledge K compensate for such lack of data. Notably, this is possible because our score is sensitive to both aspects (A1) and (A2). In other words, both the input knowledge and the task at hand have a measurable effect on the data-efficiency gain. Finally, as far as the baseline predictor is concerned (A2), we argue that the more data hungry it is, the more the data-efficiency gain will be.

4 Integration of SKI QoS metrics into PSyKI

In this section we thoroughly discuss the PSyKI system by first providing the reader with a comprehensive overview of the system, then delving into the specifics of how QoS metrics are integrated into the PSyKI library.

PSyKI—acronym for “platform for symbolic knowledge injection”—is a Python library that provides support for the injection of prior symbolic knowledge into sub-symbolic predictors by letting the users—e.g., MAS designers—choose the most adequate method with respect to the ML task to accomplish [11]. PSyKI is a tool for intelligent systems engineers who need to either experiment with already-existing SKI algorithms or invent new ones. PSyKI is public and available at github.com/psykei/psyki-python.

Currently, PSyKI can be used with predictors created by Tensorflow [61] and supports the following SKI algorithms: (1) *KBANN*, one of the first SKI algorithms proposed in literature [17]; (2) *KINS*, a structuring-based injector that integrates symbolic knowledge into a target neural networks [34]; (3) *KILL*, a constraint-based injector that affect the training of a target neural network [35].

Essentially, PSyKI is designed around the notion of *injector*, whose block diagram is shown in Fig. 4. An injector is any algorithm accepting as input a ML predictor and prior symbolic knowledge (typically logic formulæ) and producing a new predictor as output. In order to properly perform injection, injectors may require additional information, such as algorithm specific hyperparameters. The general workflow for SKI with PSyKI is

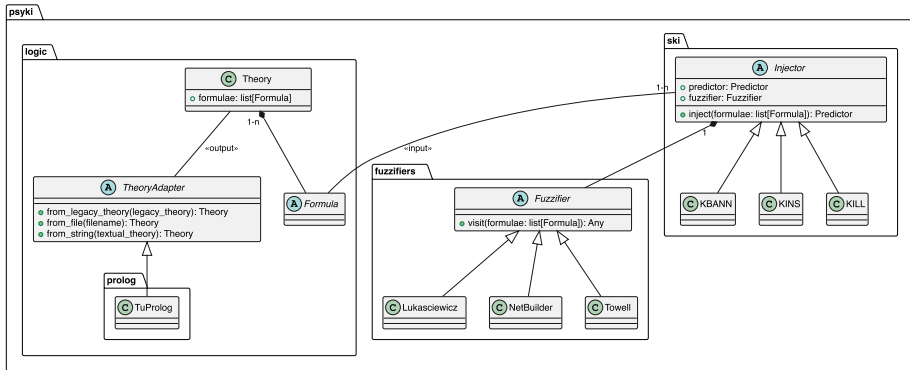


Fig. 5 Class diagram of PSyKI. Main entities are *Injector*, *Formula*, and *Fuzzifier*

compliant to the one presented in Sect. 2.1—with specific attention to parsing (Π), fuzzification (ζ) and injection (\mathcal{I}).

PSyKI supports the processing of symbolic knowledge represented via logic formulæ. Based on the sort of logic adopted, user can build an abstract syntax tree (AST) for each formula. The AST can be inspected through a *fuzzifier* via pattern visitor [62] to encode the symbolic knowledge into a sub-symbolic form (e.g. fuzzy logic functions, ad-hoc layers). The resulting sub-symbolic object can finally be used by an injector to create a new predictor. This process—denoted with ζ in Figs. 1, 2 and 3—is injector-specific; instead, the same parser Π can be used independently of the injector for logic formulæ of the same type.

The software is organised into well-separated packages and interfaces, so as to ensure extensibility towards new sorts of logics and fuzzifiers—see Fig. 5. A formula AST is represented in the software via instances of the *Formula* abstract class and its manifold subtypes (not shown in the figure)—aimed at covering the many logic-specific aspects supported by PSyKI. Ad-hoc implementations of *Formula* are included in PSyKI, one for each the logic formalism supported by our framework—currently, Prolog, Datalog, and their sub-sets –, and more may be introduced in the future by either us or other researchers by simply extending that class. The same holds for fuzzifiers (resp. injectors), i.e., subtypes of the *Fuzzifier* (resp. *Injector*) abstract class. Currently-available implementations of those class cover the KBANN [17], KINS [34], and KILL [35] injection algorithms—and the corresponding injectors as well,

However, in its original state PSyKI does not include any particular facility to assess SKI. This is why in the remainder of this paper we propose a PSyKI extension aimed at supporting engineers in need of practically assessing the effectiveness—as well as the other QoS properties discussed in this paper—of their SKI workflows.

4.1 QoS metrics implementation in PSyKI

QoS metrics are implemented as a set of classes that extend the *Metric* abstract class. Each class corresponds to a specific metric and is responsible for computing the corresponding score. Therefore, the *Metric* class provides a common interface for all metrics.

In particular, it provides two methods to compute the metric value between two predictors. The first method is `compute_during_training` and it is used to compute

the metric during the training phase of the predictors. The second method is `compute_during_inference` and it is used to compute the metric when predictors are already trained. Both methods, accept the predictors to compare as input parameters. Additional parameters can be passed to the methods to customise the computation of the metric to meet the specific needs of the user (e.g., training set, batch size, etc.).

Implemented metrics are:

1. **Memory**: memory consumption efficiency of the predictors—Equation (2);
2. **Energy**: energy consumption efficiency of the predictors—Equation (5);
3. **Latency**: latency efficiency of the predictors—Equation (7);
4. **DataEfficiency**: data efficiency of the predictors—Equation (9).

Metrics are included into the `psyki.qos` package. It is worth noting that all the metrics can be computed using any kind of predictors: no need to have one uneducated and one educated predictor. Instead, one can also compare, say, two educated predictors, or two uneducated predictors of any sort.

5 Experiments

In this section we present several experiments aimed at assessing the effectiveness of the proposed QoS metrics, as implemented in PSyKI. We first describe the experimental setup, the datasets we adopt, and the rationale behind their choice. Then, we present the results of our experiments, and we discuss them.

The design of our experiments is as follows:

1. we select three relevant classification tasks from the literature, covering different application domains, and coming with datasets of increasing cardinality;
2. for each task and its corresponding dataset D , we (1) train some uneducated neural predictor N over the data in D —of course performing train/test-set splitting—, and we (2) select some symbolic knowledge base K to be injected in N ;
3. for each uneducated predictor N we then apply SKI multiple times, one per each injection technique currently supported by PSyKI, namely KBANN, KINS, and KILL—hence attaining as many educated predictors;
4. finally, for each educated predictor \hat{N} , we compute our QoS metrics, hence comparing that \hat{N} and N w.r.t. (1) data efficiency, energy consumption, memory footprint, latency, and accuracy variation.

The rationale behind this setup is to demonstrate the effectiveness of our QoS metrics in assessing the efficiency SKI techniques of different sorts.

It is worth highlighting that the experiments presented in this section are not intended as a comprehensive evaluation of knowledge injection techniques *per se*. Instead, they aim to demonstrate the validity of the proposed QoS metrics, w.r.t. their capability of revealing *variations* in relevant efficiency metrics, as introduced by SKI. In this respect, negative values may be imputed to either the injections algorithms themselves, or to their implementation in PSyKI. In fact, the primary goal of PSyKI is to provide *correct*—despite not fully optimised—SKI techniques.

Table 2 Mapping of aggregative symbols and the four nucleotides

Symbol	Adenine	Cytosine	Guanine	Thymine
d	•		•	•
m	•	•		
r	•		•	
s		•	•	
y		•		•

Each symbol can be substituted with one base on the right that has a dot

For the sake of reproducibility, the code is public available at <https://github.com/pikalab-unibo/ski-qos-jaamas-experiments-2022>.

5.1 Datasets

We select three different datasets from the UCI repository³: BCW, PSJGS, and CI.

- *Breast cancer Wisconsin dataset (BCW)* [63] The BCW dataset contains 699 instances of breast cancer biopsy results, each with 9 features—summarising biological characteristics—and one class label. Values are integers in the range [1, 10]. The feature *BareNuclei* has 16 missing values, which are replaced with the value zero. The dataset's target variable is a binary indicator of whether a biopsy was benign (B) or malignant (M), class repartition is 458 and 241 respectively. The purpose of the dataset is to develop predictors that can accurately diagnose breast cancer based on biopsies using the information contained in the features.
- *Primate splice junction gene sequences (PSJGS)* [64] The PSJGS dataset includes information regarding gene splicing. The dataset includes 3190 instances, each representing a sequence of 60 DNA nucleotides. Each nucleotide is represented by one of the four letters A (adenine), T (thymine), C (cytosine), and G (guanine). Each sequence begins at position -30 and ends at 30, position zero is excluded.

One DNA sequence can be classified as an exon–intron (EI) boundary, an intron–exon (IE) boundary, or none (N) of them. Class frequencies are 50% for N, 25% for both EI and IE.

In addition to the four nucleotides, the dataset also includes other letters that indicate that for one specific position different nucleotides are allowed. For our experiments, we preprocess the dataset by binarising the nucleotides. In other words, each nucleotide is represented by a vector of 4 elements, where each element is 0 except for the one corresponding to the nucleotide itself, which is 1. Table 2 reports the complete binarisation of the nucleotides.

- *Census income (CI)* [65] The CI dataset contains individual information from the 1994 United States Census. The dataset contains 48,842 instances, each corresponding to one census participant. Each data row includes information such as age, education, and occupation, as well as income data about a single person. The purpose of the dataset is to predict whether an individual's annual income is greater than or less/equal than/to 50,000 USD based on their demographic information. Hence, the target variable is binary—37,155 earn less/equal than/to 50,000 USD and 11,687 earn more than that amount per year.

³ <https://archive.ics.uci.edu/ml/index.php>.

For our experiments, we convert the target *Income* to a binary output (1 if the income exceeds 50,000 USD and 0 otherwise). We also drop three features—namely *Fnlwgt*, *Education*, and *Race*—as they are irrelevant for our experiment (*Fnlwgt* is a similarity metric computed over the other features, the information provided by *Education* is already present thanks to the feature *EducationNumeric*) or possibly introduce cultural bias (*Race*). The remaining features are discretised. In particular, *CapitalGain* and *CapitalLoss* are binarised, while the remaining nominal categorical features are transformed into one-hot-encoded data.

We choose these datasets because of their increasing cardinality, which ranges from 10^2 to 10^4 . In this way, we are able to observe the scalability and robustness of our predictors and metrics in handling datasets of different volume or dimensionality. This is important to get a broader overview about the performance of the different predictors both in terms of their accuracy and in terms of the various efficiency metrics proposed in this work.

We divide each dataset into train and test sets, with a ratio of $2/3$ and $1/3$ respectively.

Finally, we attain the knowledge bases to be injected in a task-specific way. As far as the PSJGS dataset is concerned, we rely on the knowledge base described into the corresponding paper [17], which we suitably convert in Prolog form. Conversely, as far as the BCW and CI datasets are concerned, we leverage upon symbolic knowledge *extraction* [66] to automatically generate knowledge bases in Prolog form out of trained predictors. This process is better discussed into “[Appendix](#)”.

5.2 Methodology

We define and train several neural predictors, for each dataset—in particular, one uneducated network and multiple educated counterparts. We attain educated networks by applying SKI via the *KINS*, *KILL*, and *KBANN* algorithms—each one exploiting some different approach to perform knowledge injection—see Sect. 4. By constructing all such predictors, we are able to compare and evaluate their performance and their metrics on each dataset.

For each uneducated predictor, we tune the structural hyperparameters (i.e. amount of layers and neurons per layer) by using a grid search with cross-validation. Networks attained via *KBANN* are a notable exception here, as in those cases the entire architecture of the network is dictated by *KBANN*, as a function of the input knowledge. In particular, we chose to vary the number of layers (from 1 to 3) and the number of neurons per layer (10, 50, and 100). The same process of grid search with cross cross-validation is repeated for the “educated” predictors. In this way, we can ensure good hyperparameters selection—in terms of predictive performance—, while still keeping the computation time reasonable. Table 3 shows the selected hyperparameters for each dataset and predictor.

In order to tune the (hyper-)parameters of each predictor in a statistically significant way, we repeat the training 30 times, each time with different initial conditions and/or random seeds, grasping statistics about the average accuracy along the way. This lets us reduce the variability of the results and obtain a more accurate estimate of a predictor’s actual performance. The outcome of this procedure is shown in Table 4.

After calculating the average accuracy, we proceed in computing predictors’ efficiency metrics, for each dataset. In particular, we compute data efficiency, energy, memory, and latency metrics—see Sect. 3. The corresponding scores are presented in Table 4, and discussed in the following section.

Table 3 Results of a comprehensive grid search on various datasets

Dataset	Model	Layers	Neurons
BCW	Uneducated	3	[100, 10, 50]
	KBANN	3	[100, 10, 50]
	KILL	3	[100, 10, 50]
	KINS	3	[100, 10, 10]
PSJGS	Uneducated	3	[100, 10, 10]
	KBANN	3	[100, 10, 10]
	KILL	3	[100, 10, 10]
	KINS	3	[50, 10, 10]
CI	Uneducated	3	[10, 50, 50]
	KBANN	3	[10, 50, 50]
	KILL	3	[10, 50, 10]
	KINS	3	[10, 50, 10]

The models evaluated in this study are the uneducated one, and the three different educated models with KBANN, KILL, and KINS. Note that the model used by KBANN is identical to the model obtained via grid search for the uneducated model. The table provides a summary of the number of layers and neurons used in each model for each dataset

5.3 Discussion

In the following we thoroughly analyse and interpret the results of our experiments. Accordingly, we examine the columns of Table 4 from left to right.

It is worth noticing how data-efficiency scores vary hugely across predictors and datasets. We recall that a positive data-efficiency score indicates that the educated predictor is more efficient than its uneducated counterpart, whereas a negative score indicates the opposite. In general, as stated in Sect. 3, it is important to consider how data-efficiency scores can be affected by all three aspects (A1)–(A2). Thus, for instance, the high variation of this score points out the importance of selecting the most appropriate predictor for a given task (A2). For instance, the KINS-based solution has a lower data-efficiency score than the other predictors tailored on the BCW dataset. This may indicate that KINS is not the best solution for this task. In contrast, we note that the CI dataset shows positive data-efficiency scores for all three predictors, indicating that, in terms of data efficiency, an improvement is obtained by using all three SKI algorithms proposed in this work.

The second column of Table 4 shows the energy metrics for both train and test. With regard to each predictor and dataset, we mostly see negative values for this metric. Again, it is important to note that energy consumption scores can be affected by a number of factors, including input knowledge (A1), input predictor (A2), and task to be performed (A2). In most cases, the table indicates that the KBANN-based solution consumes more energy than the other predictors. In contrast, the KILL-based solution consumes significantly less energy than the other predictors. Additionally, it is important to shift the emphasis towards input knowledge (A1). As stated in Sect. 3, it is expected that the more complex the input knowledge, the more energy the educated predictor will consume during training. Hence, in terms of data efficiency, we argue that more complex knowledge may produce a gain for the educated predictor—possibly at the price of higher expenses in terms of energy consumption.

The third column of Table 4 shows the results of the memory metric. We recall that a positive value here indicates that the educated predictor consumes less memory than the

Table 4 Comparison of the performance of different models (KBANN, KILL, and KINS) respect to the uneducated one on three datasets (Breast Cancer, Splice Junction, and Census Income) in terms data efficiency, energy consumption, memory usage, latency, and accuracy

Dataset	Model	Set	Data efficiency (KB)	Energy (mWh)	Memory (FLOPs)	Latency (ms)	Accuracy (%)
BCW	Uneducated		–	–	–	–	94.53
	KBANN	Train	35.89	–1.47	3933	–1.70	95.45
		Test		–0.10			
	KILL	Train	4.09	–0.99	0	0.35	94.63
		Test		0			
	KINS	Train	–9.97	–1.22	– 559	–1.41	94.29
Test			–0.09				
PSJGS	Uneducated		–	–	–	–	93.91
	KBANN	Train	–4946.81	–4.67	–66944	–2.56	92.84
		Test		–0.22			
	KILL	Train	553.89	– 3	0	0.04	94.02
		test		0			
	KINS	Train	–954.80	–6.53	– 161,779	–4.75	93.70
Test			–0.51				
CI	Uneducated		–	–	–	–	84.63
	KBANN	Train	1653.79	–1.41	– 2468	–0.43	84.78
		Test		–0.02			
	KILL	Train	4016.90	–0.70	4200	0	84.81
		Test		0			
	KINS	Train	4263.50	–1.41	–6220	–0.44	84.77
Test			–0.02				

Note that the train accuracy is the mean of 30 runs

uneducated one. Conversely, a negative value indicates that the educated predictor consumes more memory. For example, in the case of the BCW dataset and the KBANN-based solution, the educated predictor shows a positive difference in memory consumption—which means it uses less memory than the uneducated one. In the PSJGS dataset, both KBANN- and KINS-based solutions show negative memory metrics. This suggests that, in this case, those educated predictors are more memory intensive than the uneducated one. Regarding the KILL-based solution, it often shows a memory metric of 0, indicating that there is no difference in memory between the educated and uneducated predictors.

The fourth column of Table 4 shows the latency results. Comparing the latency of educated predictors with the uneducated ones, we observe that, as far as KILL is concerned, the results between the two solutions are very similar—i.e., the metric is close to 0 in both cases. KBANN and KINS, on the other hand, have a slightly-worse latency, on all three datasets. As discussed in Sect. 3, we argue this is due to the complexity of the injected input knowledge, which can lead to negative effects in terms of latency—especially in structuring-type SKI methods, such as KBANN.

Finally, by looking at the accuracy scores—see the last column of the Table 4—, we observe that the educated and uneducated predictors show very similar results. Furthermore, the results indicate that the accuracy of all predictors on all three datasets is very

similar and consistent. In general, results suggest that all predictors perform well and can accurately predict the results of the datasets.

To conclude, in terms of data efficiency, the educated predictor generally requires less data to achieve similar accuracy than the uneducated one. This is a positive result, as it suggests that the trained predictor is able to make accurate predictions using less data, which can be a nice-to-have feature in resource-constrained settings.

As far as energy is concerned, our results show a gain in energy during the training phase for the uneducated predictor, but during the inference phase this difference is close to 0. We argue that this is due to the knowledge injection process, which in these experiments required more energy expense for the educated predictor than the uneducated one. About memory, the results are somehow mixed: the educated predictor sometimes requires more memory and sometimes less memory than the uneducated one. Finally, as far as latency is concerned, results indicate that the uneducated predictor tends to have a slightly lower latency than the educated one.

Overall, our analysis provides valuable information that can be used to understand the performance of different injection predictors on different datasets: this can be useful for evaluating a predictor with higher integrity without resorting to accuracy metrics only. This points out the importance of adopting specific metrics when evaluating knowledge injection predictors. Using these sorts of metrics in the MAS context could provide intelligent systems engineers with a critical tool for comparing different predictors and selecting the best one for a given task.

6 Conclusions

In this work we propose a set of quality-of-service (QoS) metrics for SKI mechanisms, aiming at putting MAS engineers and agents back in control of the selection of the best predictor for a given task. In particular, our metrics focus upon efficiency gains achievable through SKI. Along this line, we formally define four metrics, namely: (1) *memory footprint efficiency*—i.e., gain in terms of predictor's complexity; (2) *energy efficiency*—i.e., gain in terms of total energy required to train and deploy a sub-symbolic predictor; (3) *latency efficiency*—i.e., improvements in terms of time required for inference; and (4) *data efficiency*—i.e., improvement in terms of amount of data required to optimise a sub-symbolic predictor.

Furthermore, to support their practical exploitation, we also introduce an extension of the PSyKI library for symbolic knowledge injection, which includes a general-purpose software implementation of the metrics.

Enabled by PSyKI, we then perform a number of experiments aimed at demonstrating the effectiveness of our metrics. Overall, our experiments show that the proposed metrics can be exploited to grasp insights about whether a given SKI mechanism is actually able to improve the efficiency of a given predictor or not—according to some specific efficiency criteria among the aforementioned ones. As a by-product of our experiments we also show that the injection mechanisms currently supported by PSyKI leaves some room for improvement.

In perspective, our QoS metrics for SKI have a role to play in addressing various issues in the field of agent-oriented systems. Indeed, the design and implementation of MAS present significant challenges, such as energy consumption, latency, memory, and data efficiency. System complexity, coupled with the real-time requirements of many multi-agent applications, may lead to increased energy consumption and latency. In addition, the amount of data generated and processed by MAS can have a significant impact on their performance. Along this line, we observe that SKI approaches could reduce the amount of

computation required to process data, thus leading to reduced latency and improved energy efficiency. Similarly, it could reduce the amount of data needed for training and improve the quality of data used, thus resulting in improved performance and efficiency. The key point here is that *measuring* efficiency gains in all such regards paves the way towards the *automation* of agents' decision-making, which may then dynamically optimise their sub-symbolic components according to their goals.

Appendix: Details about the knowledge

Table 5 lists the logic rules that constitute the symbolic knowledge used in the SKI algorithms for the breast cancer dataset. A similarly-shaped knowledge has been used also in the SKI algorithms for the census income dataset.

Table 5 Knowledge used for the breast cancer dataset

Symbolic knowledge (Prolog formalism)	
<i>diagnosis</i>	<i>(BareNuclei, BlandChromatin, ClumpThickness, MarginalAdhesion, Mitoses, NormalNucleoli, SingleEpithelialCellSize, UniformityCellShape, UniformityCellSize, benign) : - UniformityCellSize < 3, NormalNucleoli > 2, BareNuclei < 1.</i>
<i>diagnosis</i>	<i>(BareNuclei, BlandChromatin, ClumpThickness, MarginalAdhesion, Mitoses, NormalNucleoli, SingleEpithelialCellSize, UniformityCellShape, UniformityCellSize, malignant) : - UniformityCellSize < 3, NormalNucleoli > 2, BareNuclei > 1.</i>
<i>diagnosis</i>	<i>(BareNuclei, BlandChromatin, ClumpThickness, MarginalAdhesion, Mitoses, NormalNucleoli, SingleEpithelialCellSize, UniformityCellShape, UniformityCellSize, benign) : - UniformityCellSize > 3, BareNuclei < 0.</i>
<i>diagnosis</i>	<i>(BareNuclei, BlandChromatin, ClumpThickness, MarginalAdhesion, Mitoses, NormalNucleoli, SingleEpithelialCellSize, UniformityCellShape, UniformityCellSize, malignant) : - UniformityCellSize > 3, BareNuclei > 0, BlandChromatin > 4.</i>
<i>diagnosis</i>	<i>(BareNuclei, BlandChromatin, ClumpThickness, MarginalAdhesion, Mitoses, NormalNucleoli, SingleEpithelialCellSize, UniformityCellShape, UniformityCellSize, malignant) : - UniformityCellSize > 3, BareNuclei > 0, BlandChromatin < 4, UniformityCellSize < 4, NormalNucleoli < 1.</i>
<i>diagnosis</i>	<i>(BareNuclei, BlandChromatin, ClumpThickness, MarginalAdhesion, Mitoses, NormalNucleoli, SingleEpithelialCellSize, UniformityCellShape, UniformityCellSize, benign) : - UniformityCellSize > 3, BareNuclei > 0, BlandChromatin < 4, UniformityCellSize < 4, NormalNucleoli > 1.</i>
<i>diagnosis</i>	<i>(BareNuclei, BlandChromatin, ClumpThickness, MarginalAdhesion, Mitoses, NormalNucleoli, SingleEpithelialCellSize, UniformityCellShape, UniformityCellSize, malignant) : - UniformityCellSize > 3, BareNuclei > 0, BlandChromatin < 4, UniformityCellSize > 4, MarginalAdhesion > 1.</i>
<i>diagnosis</i>	<i>(BareNuclei, BlandChromatin, ClumpThickness, MarginalAdhesion, Mitoses, NormalNucleoli, SingleEpithelialCellSize, UniformityCellShape, UniformityCellSize, malignant) : - UniformityCellSize > 3, BareNuclei > 0, BlandChromatin < 4, UniformityCellSize > 4, MarginalAdhesion < 1, NormalNucleoli < 3.</i>
<i>diagnosis</i>	<i>(BareNuclei, BlandChromatin, ClumpThickness, MarginalAdhesion, Mitoses, NormalNucleoli, SingleEpithelialCellSize, UniformityCellShape, UniformityCellSize, benign) : - UniformityCellSize > 3, BareNuclei > 0, BlandChromatin < 4, UniformityCellSize > 4, MarginalAdhesion < 1, NormalNucleoli > 3.</i>
<i>diagnosis</i>	<i>(BareNuclei, BlandChromatin, ClumpThickness, MarginalAdhesion, Mitoses, NormalNucleoli, SingleEpithelialCellSize, UniformityCellShape, UniformityCellSize, benign) : - NormalNucleoli < 2, BareNuclei < 4.</i>
<i>diagnosis</i>	<i>(BareNuclei, BlandChromatin, ClumpThickness, MarginalAdhesion, Mitoses, NormalNucleoli, SingleEpithelialCellSize, UniformityCellShape, UniformityCellSize, malignant) : - BareNuclei > 4, SingleEpithelialCellSize < 1.</i>
<i>diagnosis</i>	<i>(BareNuclei, BlandChromatin, ClumpThickness, MarginalAdhesion, Mitoses, NormalNucleoli, SingleEpithelialCellSize, UniformityCellShape, UniformityCellSize, benign) : - SingleEpithelialCellSize > 1.</i>

Acknowledgements This paper was partially supported by the CHIST-ERA IV project CHIST-ERA-19-XAI-005, co-funded by EU and the Italian MUR (Ministry for University and Research).

Author contributions All authors wrote and reviewed the manuscript. MM wrote and supervised the experiments. GC prepared the figures.

Funding Open access funding provided by Alma Mater Studiorum - Università di Bologna within the CRUI-CARE Agreement.

Declarations

Competing interests Partial financial support was received from CHIST-ERA.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Otter, D. W., Medina, J. R., & Kalita, J. K. (2021). A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, 32(2), 604–624. <https://doi.org/10.1109/TNNLS.2020.2979670>
2. Nassif, A. B., Shahin, I., Attili, I. B., Azzeh, M., & Shaalan, K. (2019). Speech recognition using deep neural networks: A systematic review. *IEEE Access*, 7, 19143–19165. <https://doi.org/10.1109/ACCESS.2019.2896880>
3. Zhao, Z.-Q., Zheng, P., Xu, S.-T., & Wu, X. (2019). Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems*, 30(11), 3212–3232. <https://doi.org/10.1109/TNNLS.2018.2876865>
4. Agiollo, A., & Omicini, A. (2022). GNN2GNN: Graph neural networks to generate neural networks. In J. Cussens & K. Zhang (Eds.), *Uncertainty in artificial intelligence. Proceedings of machine learning research*, vol. 180, pp. 32–42. ML Research Press, Maastricht, The Netherlands. Proceedings of the thirty-eighth conference on uncertainty in artificial intelligence, UAI 2022, 1–5 August 2022, Eindhoven, The Netherlands. <https://proceedings.mlr.press/v180/agiollo22a.html>
5. Esteva, A., Chou, K., Yeung, S., Naik, N., Madani, A., Mottaghi, A., Liu, Y., Topol, E., Dean, J., & Socher, R. (2021). Deep learning-enabled medical computer vision. *NPJ Digital Medicine*, 4(1), 1–9. <https://doi.org/10.1038/s41746-020-00376-2>
6. Grigorescu, S. M., Trasnea, B., Cocias, T. T., & Macesanu, G. (2020). A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3), 362–386. <https://doi.org/10.1002/rob.21918>
7. Calegari, R., Ciatto, G., Mascardi, V., & Omicini, A. (2021). Logic-based technologies for multi-agent systems: A systematic literature review. *Autonomous Agents and Multi-Agent Systems*, 35(1), 1–1167. <https://doi.org/10.1007/s10458-020-09478-3>. Collection “Current Trends in Research on Software Agents and Agent-Based Software Development”.
8. Kakas, A. C., & Sadri, F. (Eds.). (2002). *Computational logic: Logic programming and beyond, essays in honour of Robert A. Kowalski, part I. Lecture Notes in Computer Science* (Vol. 2407). New York: Springer. <https://doi.org/10.1007/3-540-45628-7>
9. Diligenti, M., Roychowdhury, S., & Gori, M. (2017) Integrating prior knowledge into deep learning. In *2017 16th IEEE international conference on machine learning and applications (ICMLA)* (pp. 920–923). <https://doi.org/10.1109/ICMLA.2017.00-37>
10. Xu, J., Zhang, Z., Friedman, T., Liang, Y., & den Broeck, G. V. (2018). A semantic loss function for deep learning with symbolic knowledge. In: Dy, J., Krause, A. (Eds.), *35th International Conference*

- on *Machine Learning (ICML 2018)*. Proceedings of Machine Learning Research (PLMR), vol. 80, pp. 5502–5511. Stockholmsmässan, Stockholm, Sweden. <https://proceedings.mlr.press/v80/xu18h.html>
11. Magnini, M., Ciatto, G., & Omicini, A. (2022). On the design of PSyKI: A platform for symbolic knowledge injection into sub-symbolic predictors. In D. Calvaresi, A. Najjar, M. Winikoff, & K. Främling (Eds.), *Explainable and transparent AI and multi-agent systems. Lecture Notes in Computer Science* (Vol. 13283, pp. 90–108. Springer, Cham, Switzerland. Chap. 6. 4th International Workshop, EXTRAAMAS 2022, Virtual Event, Revised Selected Papers. https://doi.org/10.1007/978-3-031-15565-9_6
 12. Calegari, R., Ciatto, G., & Omicini, A. (2020). On the integration of symbolic and sub-symbolic techniques for XAI: A survey. *Intelligenza Artificiale*, 14(1), 7–32. <https://doi.org/10.3233/IA-190036>
 13. Agiollo, A., Rafanelli, A., & Omicini, A. (2022). Towards quality-of-service metrics for symbolic knowledge injection. In A. Ferrando & V. Mascardi (Eds.), *WOA 2022—23rd Workshop “From Objects to Agents”*. *CEUR Workshop Proceedings* (Vol. 3261, pp. 30–47). <http://ceur-ws.org/Vol-3261/paper3.pdf>
 14. Smullyan, R. M. (1968). *First-order logic*. New York: Springer.
 15. Besold, T. R., d’Avila Garcez, A. S., Bader, S., Bowman, H., Domingos, P. M., Hitzler, P., et al. (2017). Neural-symbolic learning and reasoning: A survey and interpretation. CoRR abs/1711.03902 [arxiv: 1711.03902](https://arxiv.org/abs/1711.03902)
 16. Xie, Y., Xu, Z., Meel, K. S., Kankanhalli, M. S., & Soh, H. (2019). Embedding symbolic knowledge into deep networks. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. B. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems. 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8–14, 2019, Vancouver, BC, Canada*, pp. 4235–4245. <https://proceedings.neurips.cc/paper/2019/hash/7b66b4fd401a271a1c7224027ce111bc-Abstract.html>
 17. Towell, G. G., Shavlik, J. W. & Noordewier, M. O. (1990). Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings of the 8th national conference on artificial intelligence* (pp. 861–866)
 18. Tresp, V., Hollatz, J. & Ahmad, S. (1992) Network structuring and training using rule-based knowledge. *Advances in Neural Information Processing Systems*, 5, 871-878
 19. d’Avila Garcez, A. S., & Zaverucha, G. (1999). The connectionist inductive learning and logic programming system. *Applied Intelligence*, 11(1), 59–77. <https://doi.org/10.1023/A:1008328630915>
 20. d’Avila Garcez, A. S., & Gabbay, D. M. (2004). Fibring neural networks. In D. L. McGuinness & G. Ferguson (Eds.), *Proceedings of the nineteenth national conference on artificial intelligence, sixteenth conference on innovative applications of artificial intelligence, July 25–29, San Jose, California, USA* (pp. 342–347). AAAI Press/The MIT Press. <http://www.aaai.org/Library/AAAI/2004/aaai04-055.php>
 21. Bader, S., d’Avila Garcez, A. S., & Hitzler, P. (2005). Computing first-order logic programs by fibring artificial neural networks. In I. Russell, & Z. Markov (Eds.), *Proceedings of the eighteenth international florida artificial intelligence research society conference* (pp. 314–319). Clearwater Beach, FL: AAAI Press. <http://www.aaai.org/Library/FLAIRS/2005/flairs05-052.php>
 22. Chang, M., Ratniov, L., & Roth, D. (2007). Guiding semi-supervision with constraint-driven learning. In J. A. Carroll, A. van den Bosch, & A. Zaenen (Eds.), *ACL 2007, proceedings of the 45th annual meeting of the association for computational linguistics*, June 23–30, Prague, Czech Republic. <https://aclanthology.org/P07-1036/>
 23. Nickel, M., Tresp, V., Krieger, H.-P. (2011). A three-way model for collective learning on multi-relational data. *ICML, 11*, 809–816. <https://icml.cc/2011/papers/438icmlpaper.pdf>
 24. Chang, K.-W., Yih, W.-t., Yang, B., & Meek, C. (2014). Typed tensor decomposition of knowledge bases for relation extraction. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1568–1579).
 25. Guo, S., Wang, Q., Wang, L., Wang, B., & Guo, L. (2016). Jointly embedding knowledge graphs and logical rules. In J. Su, X. Carreras & K. Duh (Eds.), *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, Austin, Texas, USA, November 1–4, 2016, pp. 192–202. <https://doi.org/10.18653/v1/d16-1019>
 26. Hu, Z., Ma, X., Liu, Z., Hovy, E. H., & Xing, E. P. (2016). Harnessing deep neural networks with logic rules. In *Proceedings of the 54th annual meeting of the association for computational linguistics, ACL 2016*, August 7–12, 2016, Berlin, Germany, Volume 1: Long Papers. <https://doi.org/10.18653/v1/p16-1228>
 27. Hu, Z., Yang, Z., Salakhutdinov, R., & Xing, E. P. (2016). Deep neural networks with massive learned knowledge. In J. Su, X. Carreras, & K. Duh (Eds.), *Proceedings of the 2016 conference on empirical methods in natural language processing, EMNLP 2016*, Austin, Texas, USA, November 1–4, 2016, pp. 1670–1679. <https://doi.org/10.18653/v1/d16-1173>

28. Nickel, M., Rosasco, L., & Poggio, T. (2016). Holographic embeddings of knowledge graphs. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 30). <https://doi.org/10.1609/aaai.v30i1.10314>
29. Diligenti, M., & Gori, M., & Saccà, C. (2017). *Semantic-based regularization for learning and inference* (Vol. 244, pp. 143–165). <https://doi.org/10.1016/j.artint.2015.08.011>
30. Evans, R., & Grefenstette, E. (2018). Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61, 1–64. <https://doi.org/10.1613/jair.5714>
31. Sourek, G., Aschenbrenner, V., Zelezný, F., Schockaert, S., & Kuzelka, O. (2018). Lifted relational neural networks: Efficient learning of latent relational structures. *Journal of Artificial Intelligence Research*, 62, 69–100. <https://doi.org/10.1613/jair.1.11203>
32. Sun, Z., Deng, Z.-H., Nie, J.-Y. & Tang, J. (2019). Rotate: Knowledge graph embedding by relational rotation in complex space. CoRR [arXiv:1902.10197](https://arxiv.org/abs/1902.10197)
33. Badreddine, S., d'Avila Garcez, A., Serafini, L., & Spranger, M. (2022). Logic tensor networks. *Artificial Intelligence*, 303, 103649. <https://doi.org/10.1016/j.artint.2021.103649>
34. Magnini, M., Ciatto, G., & Omicini, A. (2022). KINS: Knowledge injection via network structuring. In R. Calegari, G. Ciatto, & A. Omicini (Eds.), *CILC 2022—Italian Conference on Computational Logic CEUR Workshop Proceedings* (Vol. 3204, pp. 254–267). http://ceur-ws.org/Vol-3204/paper_25.pdf
35. Magnini, M., Ciatto, G., & Omicini, A. (2022). A view to a KILL: Knowledge injection via lambda layer. In A. Ferrando & V. Mascardi (Eds.), *WOA 2022—23rd workshop “From objects to agents”. CEUR workshop proceedings* (Vol. 3261, pp. 61–76). <http://ceur-ws.org/Vol-3261/paper5.pdf>
36. Marra, G., Giannini, F., Diligenti, M., & Gori, M. (2019). LYRICS: A general interface layer to integrate logic inference and deep learning. In U. Brefeld, E. Fromont, A. Hotho, A. J. Knobbe, M. H. Maathuis, & C. Robardet (Eds.), *Machine learning and knowledge discovery in databases—European conference, ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, proceedings, Part II. Lecture Notes in Computer Science*, vol. 11907, pp. 283–298. Springer. https://doi.org/10.1007/978-3-030-46147-8_17
37. Ruder, S. (2016). An overview of gradient descent optimization algorithms. CoRR [abs/1609.04747](https://arxiv.org/abs/1609.04747)
38. Demeester, T., Rocktäschel, T. & Riedel, S. (2016). Lifted rule injection for relation embeddings. In J. Su, X. Carreras, & K. Duh (Eds.), *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, Austin, Texas, USA, November 1–4, 2016, pp. 1389–1399 <https://doi.org/10.18653/v1/d16-1146>
39. França, M. V. M., Zaverucha, G., & Garcez, A. S. D. (2014). Fast relational learning using bottom clause propositionalization with artificial neural networks. *Machine Learning*, 94(1), 81–104. <https://doi.org/10.1007/s10994-013-5392-1>
40. Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., & De Raedt, L. (2021). Neural probabilistic logic programming in deepproblog. *Artificial Intelligence*, 298, 103504. <https://doi.org/10.1016/j.artint.2021.103504>
41. Bordes, A., Usunier, N., García-Durán, A., Weston, J., & Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. In C. J. C. Burges, L. Bottou, Z. Ghahramani, & K. Q. Weinberger (Eds.), *Proceedings of 27th annual conference on neural information processing systems (NeurIPS)*, Lake Tahoe, Nevada, United States, December 5–8, 2013, pp. 2787–2795. <https://proceedings.neurips.cc/paper/2013/hash/1cecc7a77928ca8133fa24680a88d2f9-Abstract.html>
42. Wang, Q., Wang, B., & Guo, L. (2015). Knowledge base completion using embeddings and rules. In Q. Yang, & M. J. Wooldridge (Eds.), *Proceedings of the 24th international joint conference on artificial intelligence (IJCAI)*, Buenos Aires, Argentina, July 25–31, pp. 1859–1866. AAAI Press. <http://ijcai.org/Abstract/15/264>
43. Liu, Q., Jiang, H., Ling, Z., Wei, S., & Hu, Y. (2016). Probabilistic reasoning via deep learning: Neural association models. CoRR [abs/1603.07704](https://arxiv.org/abs/1603.07704) [arxiv:1603.07704](https://arxiv.org/abs/1603.07704)
44. Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., Melo, G. D., Gutierrez, C., Kirrane, S., Gayo, J. E. L., Navigli, R., Neumaier, S., et al. (2021). Knowledge graphs. *ACM Computing Surveys (CSUR)*, 54(4), 1–37.
45. Ahmad, T., Zhang, D., Huang, C., Zhang, H., Dai, N., Song, Y., & Chen, H. (2021). Artificial intelligence in sustainable energy industry: Status quo, challenges and opportunities. *Journal of Cleaner Production*, 289, 125834. <https://www.sciencedirect.com/science/article/abs/pii/S0959652621000548>
46. Piedmont, R. L. (2014). In A. C. Michalos (Ed.), *Bias, statistical* (pp. 382–383). Dordrecht: Springer. https://doi.org/10.1007/978-94-007-0753-5_2865
47. Liberis, E., Dudziak, L., & Lane, N. D. (2021). μ mas: Constrained neural architecture search for micro-controllers. In E. Yoneki, & P. Patras (Eds.), *EuroMLSys@EuroSys 2021, proceedings of the 1st workshop on machine learning and Systems virtual event*, Edinburgh, Scotland, UK, 26 April, 2021, pp. 70–79. ACM. <https://doi.org/10.1145/3437984.3458836>

48. Agiollo, A., Ciatto, G., & Omicini, A. (2021). Shallow2Deep: Restraining neural networks opacity through neural architecture search. In D. Calvaresi, A. Najjar, M. Winikoff, K. Främling (Eds.), *Explainable and transparent AI and multi-agent systems. Third international workshop, EXTRAAMAS 2021, virtual event, May 3–7, 2021, Revised Selected Papers. Lecture Notes in Computer Science* (Vol. 12688, pp. 63–82). Cham, Switzerland: Springer. https://doi.org/10.1007/978-3-030-82017-6_5. http://link.springer.com/10.1007/978-3-030-82017-6_5
49. Agiollo, A., & Omicini, A. (2021). Load classification: A case study for applying neural networks in hyper-constrained embedded devices. *Applied Sciences*. <https://doi.org/10.3390/app112411957>. Special Issue “Artificial Intelligence and Data Engineering in Engineering Applications”.
50. Kang, D., Kim, E., Bae, I., Egger, B., & Ha, S. (2018). C-good: C-code generation framework for optimized on-device deep learning. In *2018 IEEE/ACM international conference on computer-aided design (ICCAD)* (pp. 1–8). <https://doi.org/10.1145/3240765.3240786>
51. Wu, B., Wan, A., Yue, X., Jin, P., Zhao, S., Golmant, N., et al. (2018) Shift: A Zero FLOP, Zero Parameter Alternative to Spatial Convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 9127–9135). <https://doi.org/10.1109/CVPR.2018.00951>
52. Liberis, E., Dudziak, Ł., & Lane, N. D. (2021) μ mas: Constrained neural architecture search for micro-controllers. In *Proceedings of the 1st workshop on machine learning and systems* (pp. 70–79).
53. Huang, G., Liu, S., van der Maaten, L., & Weinberger, K. Q. (2018). Condensenet: An efficient densenet using learned group convolutions. In *2018 IEEE conference on computer vision and pattern recognition, CVPR 2018, Salt Lake City, UT, USA, June 18–22, Computer Vision Foundation/IEEE Computer Society*, pp. 2752–2761. <https://doi.org/10.1109/CVPR.2018.00291>
54. Cheng, H., Zhang, T., Yang, Y., Yan, F., Teague, H., Chen, Y., et al. (2019). Msnet: Structural wired neural architecture search for internet of things. In *2019 IEEE/CVF international conference on computer vision workshops, ICCV Workshops 2019, Seoul, Korea (South), October 27–28*, pp. 2033–2036. IEEE. <https://doi.org/10.1109/ICCVW.2019.00254>
55. Wu, B. (2019) Efficient deep neural networks. CoRR abs/1908.08926 [arxiv:1908.08926](https://arxiv.org/abs/1908.08926)
56. Savazzi, S., Nicoli, M., Bennis, M., Kianoush, S., & Barbieri, L. (2021). Opportunities of federated learning in connected, cooperative, and automated industrial systems. *IEEE Communications Magazine*, 59(2), 16–21.
57. Hou, W., Fu, M., Zhang, H., & Wu, Z. (2017). Consensus conditions for general second-order multi-agent systems with communication delay. *Automatica*, 75, 293–298. <https://doi.org/10.1016/j.automatica.2016.09.042>
58. Shumailov, I., Zhao, Y., Bates, D., Papernot, N., Mullins, R. D., & Anderson, R. (2021). Sponge examples: Energy-latency attacks on neural networks. In *IEEE European symposium on security and privacy, EuroS & P 2021, Vienna, Austria, September 6–10*, pp. 212–231. IEEE. <https://doi.org/10.1109/EuroSP51992.2021.00024>
59. Deng, J., & Ren, F. (2021) A survey of textual emotion recognition and its challenges. *IEEE Transactions on Affective Computing*, 14, 49–67. <https://doi.org/10.1109/TAFFC.2021.3053275>
60. Sanchez-Iborra, R., & Skarmeta, A. F. (2020). TinyML-enabled frugal smart objects: Challenges and opportunities. *IEEE Circuits and Systems Magazine*, 20(3), 4–18. <https://doi.org/10.1109/MCAS.2020.3005467>
61. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., et al. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. <https://www.tensorflow.org/>
62. Gamma, E., Helm, R., Johnson, R. E., & Vlissides, J.M. (1993). Design patterns: Abstraction and reuse of object-oriented design. In O. Nierstrasz (Ed.), *ECOOP’93—Object-oriented programming, 7th European conference, Kaiserslautern, Germany, July 26–30, 1993, Proceedings. Lecture Notes in Computer Science* (Vol. 707, pp. 406–431). Springer. https://doi.org/10.1007/3-540-47910-4_21
63. Wolberg, W. H. (1992). UCI machine learning repository. <http://archive.ics.uci.edu/ml>
64. Towell, G. G., & Shavlik, J. W. (1994). Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1–2), 119–165. [https://doi.org/10.1016/0004-3702\(94\)90105-8](https://doi.org/10.1016/0004-3702(94)90105-8)
65. Dua, D., & Graff, C. (2017). UCI machine learning repository. <http://archive.ics.uci.edu/ml>
66. Sabbatini, F., Ciatto, G., Calegari, R., & Omicini, A. (2022). Symbolic knowledge extraction from opaque ML predictors in PSyKE: Platform design & experiments. *Intelligenza Artificiale*, 16(1), 27–48. <https://doi.org/10.3233/IA-210120>

Authors and Affiliations

Andrea Agiollo¹ · Andrea Rafanelli^{2,3} · Matteo Magnini¹ · Giovanni Ciatto¹ · Andrea Omicini¹

✉ Matteo Magnini
matteo.magnini@unibo.it

Andrea Agiollo
andrea.agiollo@unibo.it

Andrea Rafanelli
andrea.rafanelli@phd.unipi.it

Giovanni Ciatto
giovanni.ciatto@unibo.it

Andrea Omicini
andrea.omicini@unibo.it

¹ Dipartimento di Informatica – Scienza e Ingegneria (DISI), Alma Mater Studiorum—Università di Bologna, Cesena, Italy

² Dipartimento di Informatica, Università di Pisa, Pisa, Italy

³ Dipartimento di Informatica – Scienza e Ingegneria e Matematica (DISIM), Università dell’Aquila, L’Aquila, Italy