Science & Education
https://doi.org/10.1007/s11191-024-00504-4

SI: EPISTEMIC INSIGHT & ARTIFICIAL INTELLIGENCE

Epistemic Insights as Design Principles for a Teaching-Learning Module on Artificial Intelligence

Eleonora Barelli[1,4] · **Michael Lodi**[2] · **Laura Branchetti**[3] · **Olivia Levrini**[1]

Accepted: 2 January 2024

Abstract
In a historical moment in which Artificial Intelligence and machine learning have become within everyone's reach, science education needs to find new ways to foster "AI literacy." Since the AI revolution is not only a matter of having introduced extremely performant tools but has been determining a radical change in how we conceive and produce knowledge, not only technical skills are needed but instruments to engage, cognitively, and culturally, with the epistemological challenges that this revolution poses. In this paper, we argue that epistemic insights can be introduced in AI teaching to highlight the differences between three paradigms: the imperative procedural, the declarative logic, and the machine learning based on neural networks (in particular, deep learning). To do this, we analyze a teaching-learning activity designed and implemented within a module on AI for upper secondary school students in which the game of tic-tac-toe is addressed from these three alternative perspectives. We show how the epistemic issues of opacity, uncertainty, and emergence, which the philosophical literature highlights as characterizing the novelty of deep learning with respect to other approaches, allow us to build the scaffolding for establishing a dialogue between the three different paradigms.

1 Introduction

In the era of Artificial Intelligence (AI), a radical change has been taking place. Undoubtedly, a change in the impact of science and technology on society: not only the applications of machine learning (ML) have reached people's life and behavior (Rudin & Wagstaff, 2014) but they often generate strong emotional reactions, from a deep curiosity to fears and negative attitudes (O'Neil, 2016). This has become more evident in the last months, with

✉ Eleonora Barelli
eleonora.barelli2@unibo.it

[1] Department of Physics and Astronomy "Augusto Righi", Alma Mater Studiorum - University of Bologna, Bologna, Italy

[2] Department of Computer Science and Engineering, Alma Mater Studiorum - University of Bologna, Bologna, Italy

[3] Department of Mathematics "Federigo Enriques", University of Milan, Milan, Italy

[4] Present Address: IFAB - International Foundation Big Data and Artificial Intelligence for Human Development, Bologna, Italy

Published online: 23 February 2024

large language models embedded in chatbots like ChatGPT and picture-generative models like DALL·E 2 being easily available without any requirement of specific literacy. A recent review by Leiter et al., (2023) studied how tweets and scientific papers perceived or discussed about ChatGPT: even though it is generally viewed as of high quality, the excitement decreased since its debut, opening up many concerns specifically in the educational fields, in which it is seen as a threat to academic integrity that can foster dishonesty (Ventayen, 2023).

At the epistemological level, the philosophy of computer science has started to investigate the implications of new data-driven approaches and there are authors who have seen in these changes a "paradigm shift" (Hey et al., 2009; Kitchin, 2014). The most radical supporters of the paradigm shift claimed for a new purely inductive modus operandi in doing science (Prensky, 2009). According to these drastic views, the data are supposed to speak for themselves being free of theory. The critics of this radical form of empiricism argue that each system, including the ML ones, is designed to capture certain kinds of data (Berry, 2011; Leonelli, 2012) and, then, the results cannot be considered as free from theory, neither can they simply ask from themselves free of human bias (Gould, 1981). Between extreme positions that characterize the debate, it became very common lately to consider a "data-driven science" (Kitchin, 2014) as a hybrid combination of abductive, inductive, and deductive approaches.

Beyond the epistemological debate about how to define the potential ongoing "paradigm shift," this issue is of outstanding importance also for research in science education. On one hand, big data and new data analytics such as ML techniques based on neural networks (in particular, deep learning) and generative methods are disruptive innovations which are reconfiguring in many instances our relationship with technology. On the other hand, these rapid changes have only recently been supported by an educational and cultural reflection on the implications of the unfolding revolution that touches not only technical and conceptual issues but also epistemological ones.

In this paper, we aim at contributing to this issue, by answering the following research question: *Which epistemological insights should and can be introduced to upper secondary-school students to make them aware of the AI and ML revolution?* To do that, we frame our investigation within the perspective of epistemic insights, conceptualized by Billingsley as "knowledge about knowledge with a focus on knowledge about disciplines and how they interact" (Billingsley et al., 2018). For Billingsley and colleagues, interdisciplinarity is key to making knowledge learned at school "epistemically insightful." Indeed, fragmentation of teaching in subjects leads often to a disconnected learning experience that hinders meaningful connections between key concepts and knowledge across curriculum disciplines (Billingsley, 2017). On the opposite, there are central issues that can only be addressed by multiple disciplinary perspectives: Billingsley formulates them as "big questions" that concern the nature of reality and human personhood (Billingsley et al., 2018). We found this framework suitable for our goal because, on one side, the paradigm shift brought by AI raises meaningful questions, e.g., about the relationship between humans and machines and about the meaning of intelligence, knowledge, and creativity, and, on the other, intersects several disciplines, from STEM domains (like computer science, physics, and mathematics) to humanities (e.g., philosophy and arts).

To answer our research question, we analyze in this paper part of a module on AI targeted to upper secondary-school students, designed within the Erasmus+ project I SEE (https://iseeproject.eu/.) and refined within the Horizon2020 project FEDORA* (https://www.fedora-project.eu/). Both projects have developed and implemented—several times and in different contexts—teaching-learning modules on advanced STEM topics, always valuing the epistemological and cultural revolutions that issues like AI, climate change, and quantum computers embed. In particular, the epistemological layer is the locus where we positioned the key design principles that guided the educational reconstruction (Duit et al.,

2012) of the basic disciplinary content knowledge that we addressed in the AI module, the object of this article. More specifically, the epistemic issues of *opacity*, *uncertainty*, and *emergence* have been analyzed to build the scaffolding for introducing and critically comparing three very different paradigms within AI: the procedural imperative, the declarative logic, and the deep learning. In this paper, we analyze how specific questions, grounded in the literature on the epistemology of AI, can promote epistemic insights starting from the presentation of simple examples like the game of tic-tac-toe (also known as "noughts and crosses" or "Xs and Os") addressed with the three aforementioned paradigms.

The paper is structured as follows. In Section 2, we present a literature review on the state of the art of education to AI. In Section 3, we analyze the body of literature in the field of philosophy to identify key epistemological issues related to AI and, in particular, ML and deep learning. Then, in Section 4, we present the design principles that guided the design of the module on AI, focusing in particular on the epistemic insights that we want to foster. Section 5 is devoted to illustrating the articulation of the module in preparation for the work we do in Section 6: here, we present the epistemic insights raised by the imperative procedural, declarative logic, and deep learning paradigms when confronting the same operational task, i.e., how to make a computer play tic-tac-toe.

2 Literature Review: State of the Art of Education to AI

In the last years, massive attention has been devoted, in the context of computer science education (CSEd), to teaching computational thinking (Lodi & Martini, 2021).

Computational thinking was informally defined by Wing (2006) as "thinking like a computer scientist" to solve problems and, more formally, as "the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent" (Wing, 2011).

Zeng (2013) noted that this approach shows especially "logic- and algorithm-based perspectives." He proposed to also consider what he called "AI thinking," to "leverage knowledge bases and case bases in problem solving, capture and reason about commonsense, enable processing of semantics and contexts, and deal with unstructured data, among others" (Zeng, 2013, p. 3).

The main focus of research regarding AI and education has been applying AI tools to support education (*AI in education*). However, one of the aims of CS education is to make computing systems appear not to be magic by explaining (some of) their underlying mechanisms. Only an "undercurrent" dealt with teaching how AI works to K-12 students (*education to AI*). Since the 1970s, this research followed the AI topics that were "hot" in the different decades, focusing, for example, on robots navigating the environment, natural-language processing, and expert systems. However, thanks to the explosion of data-driven techniques like ML with neural networks, it is foreseen that "the next frontier in computer science education research is how to teach artificial intelligence" (Tedre et al., 2021, p. 110559).

A very recent systematic review by Sanusi et al. (2022) analyzed 43 papers related to teaching and learning ML in K-12 education. While they recognize that some ML activities have been introduced in K-12, especially in high school, they also recognize a scattered situation: small sample sizes, limited and unstructured learning material and resources, lack of teacher training, lack of structured assessment tools, a narrow focus on ML classification

tasks, need for more integration with other subjects' domains, and lack of focus on societal and ethical implications.

Tedre et al. (2021) conducted a scoping review, highlighting several educational changes that are taking place in CS education according to the works focusing on the introduction of ML in K-12 education. Following Tedre and colleagues' work, we summarize the most relevant in our context.

From a pedagogical and methodological point of view, some important aspects emerged from the reviewed works.

First of all, the applications in which ML thrives are often media applications (e.g., image, music, and speech recognition), where it is easier to collect a lot of data and reach the desired goal by training a neural network rather than write a traditional program (for example, because precise algorithms or mathematical modelings of the phenomena are lacking or extremely complicated). Many educational initiatives are in fact media-centered: they fit well in modern pedagogies focused on playful exploration and creative learning and allow students to reach non-trivial motivating results (i.e., programs that show complex behaviors) more easily.

Next, ML can favor a STEAM integration by making students solve problems using "rich, real-world data," potentially favoring a "shift from rule-based (deductive, positivist) reasoning towards data-driven (inductive, falsificationist) reasoning—which is well in line with how natural sciences work" (Tedre et al., 2021, p. 110567).

The choice of the context in which to teach ML is crucial. Many ML in K-12 initiatives recognize that features of services that kids and young people use are based on ML, and ML will affect their lives and future work dramatically. Moreover, many scholars argue that ML education also needs the ethics of AI. ML education can and should discuss topics like "privacy, surveillance, job losses, misinformation, diversity, algorithmic bias, transferability, and accountability."

By asking students to train ML models with a lot of data, the curation, cleaning, and labeling of data have become important learning objectives. The goal should be, of course, not to master the technical details or to achieve professional results (often hindered by poor-quality data like blurred photos or noisy recordings), but to better understand one of the big ideas of AI—"Computers can learn from data" (Touretzky et al., 2019)—and its implications (e.g., the impact, on the results, of the sample size or the biases introduced by the sample choice).

Education to AI has to deal, at some point, with a language through which the machine is programmed. Programming languages' syntax has always been a source of cognitive load for students, so traditionally CS Education developed alternatives, like the recent trend on block-based languages. Some ML teaching initiatives maintain the traditional (textual or block-based) approach, using traditional programming to teach ML topics. Other initiatives hid completely the programming syntax (e.g., with a web-based visual interface) or changed the focus from programming to the design of the neural network structure, showing, for example, nodes, layers, and connections of a deep neural network.

The biggest impact of the paradigm shift caused by ML techniques is exactly how one can design and implement the solution to a problem with an information processing agent.

Traditionally, an algorithm is seen as a step-by-step method to solve a problem, with "discrete, deterministic, unambiguous atomic operations" (Tedre et al., 2021, p. 110565). In a neural network, it is impossible to find these "steps" in the solution. Moreover, most ML applications hide what happens inside the network. When designing, training, testing, and using an ML model, computer scientists do not think about algorithmic steps but how to design a dataset that creates desired user behaviors: there is a shift from coding

to "teachable machines." Therefore, students must "understand how ML 'reasons' using models and representations built from data, and not rules coded in the system" (ibidem). However, Tedre and colleagues recognize a need to teach traditional CT as well, because real applications are always a combination of traditional programming and ML.

Traditional programming can be called a *glass box* in the sense that the "flow of program execution, changes in values of variables, and everything else a program does are all hand-coded in the program, and the program flow can be tracked, visualized, and paused at any point to examine the program state at any given step of execution" (ibidem). In ML, and especially in the sub-field of deep learning, we can see different levels of the so-called black boxes, i.e., objects that "return outputs [...] by running internal processes that are incomprehensible by human beings" (Carabantes, 2020):

- In a neural network, weights and parameters are not set by hand but trained by giving the network a lot of data
- Examining the weights and parameters of a trained neural network gives no interpretation of what it does or how: this is considered one of the major challenges of AI education, and more generally for AI (see Section 3)
- Educationally, you can always decide which levels of details to show and which to hide: the more accessible, abstract, and simplified the model is, the more it hides from users, the less they learn of what happens inside. This can lead to oversimplified or inaccurate mental models of AI/ML systems, also creating in students unrealistic expectations from them (with anthropomorphizing and personification), leading to the so-called "Eliza effect," when a system appears much smarter than it is

ML has a huge impact also on one of the most common practices of programmers: debugging an ML model is very different from traditional programs. ML results are not discrete but, for example, expressed as the probability that an input belongs to a certain class. They are also "brittle", i.e., very dependent on the environment or input data. However, this can teach students about relevant ideas like overfitting, accuracy, and precision. Traditional debugging is systematic, while ML is more of a trial-and-error search for optimal parameters, features, weights, and so on.

These didactical changes and open issues require an epistemological focus, to which the following sections are dedicated.

3 Literature Review: Epistemological Issues Related to AI

The epistemological issues concerning AI emerged since the very first attempts to write a definition, on which there is nowadays very little consensus (Monett & Lewis, 2017; Nilsson, 2010). The term "Artificial Intelligence" was coined by John McCarthy in 1956, to mean the set of techniques, procedures, and methods to make machines do things that would require intelligence if they were done by humans. The reference to human intelligence started to be the benchmark to evaluate any kind of computational artifact; hence, the difficulties in conceptualizing intelligence at large, together with the multiple interpretations according to philosophical and cognitive theories, make the convergence on the meaning of AI particularly challenging (Wang, 2019).

For the purposes of this paper, in the plethora of AI disciplines, we will specifically enter the philosophical underpinnings of ML. Without any presumption of generalization

and completeness, we will introduce some concepts and issues that have been discussed from an epistemological point of view in relation to ML.

ML is a vast field, including several techniques (like learning decision trees or linear regression) with different characteristics with respect to the issues we discuss. Because of the introductory nature of the work, while still using the more general and widespread keyword "Machine Learning," we focused on, and always refer to, its most widely used set of techniques: deep learning (Russell & Norvig, 2021 p. 801) using neural networks.

Our analysis is organized around three main key issues: *opacity*, *uncertainty*, and *emergence*. As we will show, these issues encompass many others.

Opacity is the property that characterizes the aforementioned "black boxes." Jenna Burrel states that opacity appears in the field of ML in three forms: (i) opacity as intentional corporate or institutional self-protection and concealment; (ii) opacity as technical illiteracy, stemming from the current state of affairs where writing and reading code is a specialist skill; and (iii) opacity as arising from the characteristics of ML algorithms and the scale required to apply them usefully. We focus our epistemological analysis on the latter type of opacity since it refers to the mismatch between the inner conceptual structure of many ML algorithms and the demands of human-scale reasoning and styles of semantic and symbolic interpretation (Burrell, 2016). This opacity is not a matter of intentional hiding nor of technical incapability: even the designers of ML systems are unable to explain the outcomes produced by the machine. It is an inherent complexity that arises from algorithmic optimization procedures that exceed the capacity of human cognition (Desai et al., 2022). As human beings, we can store up to a certain amount of information in our brains and can reliably handle even less, our computations are slow and too prone to errors, and algorithms are extremely complex entities to be surveyed (Durán & Jongsma, 2021).

Opaque ML methods challenge our cognitive systems not only for a matter of problems' scale (too much data, too long to be processed, and too many errors). From an epistemic perspective, algorithms like artificial neural networks are *sub-symbolic*. Contrary to symbolic methods that rely on a knowledge base made by logic rules, ontologies, decision trees, planning, and reasoning which are human-readable and interpretable, sub-symbolic methods establish correlations between input and output variables that are not formulated in terms of symbols but of numbers (Ilkou & Koutraki, 2020).

To overcome the limitations of these epistemically opaque systems, which are particularly evident when they support decision-making processes in real-life settings (e.g., autonomous driving, medical diagnostic and robotics, or finance), a new field in AI has risen in the last decade: Explainable Artificial Intelligence (XAI), whose aim is to achieve machines' transparency by providing both textual and visual explanations for the outcomes obtained and the procedures carried out, hence, to foster trust in algorithms and their results (Goebel et al., 2018; Gunning & Aha, 2019; Zednik & Boelsen, 2021). In XAI, some investigators develop ML methods that are "inherently interpretable" (Rudin, 2019). Others, in contrast, address the epistemic opacity issue by developing analytic techniques to enable "post-hoc explanations" (Zednik, 2021). The latter type includes symbolic metamodeling which proposes an interpretation of machines' decisions in terms of familiar mathematical functions that can be symbolically manipulated by humans (Alaa & van der Schaar, 2019). The difference between inherent and post hoc explanations recalls the distinction that some scholars draw between "explainable" and "interpretable" methods (Tsamados et al., 2021). Explainability relates mainly to non-experts: in this context, XAI is particularly important when considering the rapidly growing number of open source and easy-to-use models and datasets, with increased numbers of non-experts who are experimenting with state-of-the-art algorithmic models, often without having the necessary technical literacy to grasp

the meaning of their results. On the other side, interpretability regards mainly experts: the methods of XAI should allow the unveiling of internal processes of ML systems in a way that technically literate people can understand them. Hence, in the view of Tsamados and colleagues, explainable methods presuppose interpretability but not vice versa.

The opacity of many classes of ML systems is connected to the second epistemologically relevant issue we want to address, which is *uncertainty*. The traditional view of computing is usually conceived as a collection of symbolic and deterministic algorithms that, because of these two features, can be verified (Shapiro et al., 2018). This approach has valued correctness and verifiability as fundamental epistemological stances of deterministic systems, as highlighted by MacKenzie (2001), and has generated a bivalent view of correctness, which assigns programs to one of two categories: either correct or incorrect (Tedre et al., 2021). As computing has evolved toward ML methods, we have observed a departure from this traditional view. For epistemologists, in ML, the emphasis is moved to effectiveness rather than correctness.

According to Valiant (2013), the goodness of an ML solution can at best be "probably approximately correct," with its goodness statistically determined. This is due to at least three factors (Hüllermeier & Waegeman, 2021; Kläs & Vollmer, 2018). The first is that ML systems involve the interactions of large numbers of components to generate complex emergent behaviors (this point is explored in detail in the next paragraph). The second reason is connected to the data that feed the algorithms, from which the machines learn: data are limited in accuracy and potentially affected by various kinds of quality issues and biases (also this point is explored in the next paragraph). A third reason for the statistical character of the output of ML methods is connected to the nature itself of tasks that are addressed through this approach, i.e., problems for which at the moment we cannot elaborate a procedure to achieve a unique correct result, but their complexity leads to uncertainty in the prediction of the outcome.

For these reasons, the dichotomic view of correctness is not suitable to address the character of the results of ML systems: more fuzzy views have to be embraced, that involve reliability and efficiency in a contextual, relative, and pragmatic view of goodness (Tedre et al., 2021).

The third epistemologically relevant issue in ML we consider is *emergence*. The word emergence is typical of complex systems that often display global overall properties that arise—emerge—from simple procedures. Considering the very well-known example of ants in a colony, biological and computational models have demonstrated that the intelligent, efficient, and cooperative behavior of the colony is the result of the simple behavior of each worker ant that searches for food and, when it finds it, goes back to the anthill marking the path as it returns, and the path stimulates other ants to forage in the same direction (van Zuylen, 2012). Emergence does not only concern the field of ML strictly speaking but it is also considered broadly when dealing with big data: indeed, in the so-called era of big data (Boyd & Crawford, 2012), the combination of data-intensive approaches and ML, "much knowledge is [...] produced autonomously by the tools scientists have made, and not directly by the scientists themselves" (Sætra, 2018, p. 509). In ML, emergence stems from one of the very first definitions of it given by Arthur Samuel who invented the word machine learning as the "field of study that gives computers the ability to learn without being explicitly programmed" (Samuel, 1959). The abilities that the machine obtains through training, e.g., the capacity to label unknown images as "cats" or "chairs," or to generate new images starting from a textual prompt, can be considered emergent properties. The knowledge that the machine acquires is not an *a priori* set competence but a property of the trained, validated, and tested algorithm that emerges from

simple local interactions and procedures coded. Many ML methods, particularly artificial neural networks, lack a central control: the processing is distributed over the network in a process of self-organization which starts from data and has learning as its result (Anegawa et al., 2023; Cilliers, 1998).

The concept of emergence and "learning from data" has allowed achieving remarkable results in the last decades but poses many challenges from an epistemological perspective. Indeed, learning from data starts from observations to achieve broader generalizations (Hammoudeh et al., 2021). Hence, if we adopted sharp distinctions between deductive and inductive approaches to knowledge, or between rationalism and empiricism, ML would situate in the inductive and empiricist categories (Harman & Kulkarni, 2007). We need to clarify that the bottom-up approach typical of ML is not common to all AI methods: indeed, many AI systems are developed with a top-down approach, in particular, in the case of expert systems that share many features with traditional programming, in which the knowledge base is provided to the machine in the form of instructions (imperative procedural approach) or of rules and statements and an inference engine deduces new statements by applying rules to the known ones (declarative logic approach).

Several scholars argue that ML methods, powered with big data availability, have given rise to a new epistemology and paradigm shifts in which data and algorithms can drive scientific discoveries and knowledge creation (Hey et al., 2009), by revolutionizing the scientific method (Prensky, 2009; Halevy et al., 2009) and eventually determining the end of theory (Anderson, 2008). Kitchin (2014) critiques the enthusiasm for these claims identifying four fallacies. First, data are not simply natural, values-free elements that can be abstracted from the world, but they are generated through a complex assemblage that actively shapes its constitution and often includes the perpetuation of biases (Ribes & Jackson, 2013). Second, theory is embedded in data: no inductive strategy of knowledge extraction from data occurs in a scientific vacuum, because it is framed by previous findings, theories, experience, and knowledge (Leonelli, 2012). Third, data cannot speak for themselves free of human bias: they are always examined through a particular lens that influences how they are interpreted, and the construction of algorithms is imbued with particular values and contextualized within a particular scientific approach. Fourth, it is not true that an understanding of statistics is enough to allow data interpretation, without the need for any domain-specific knowledge: subject matter experts are still needed to assess the results of the work, especially when we deal with sensitive data about human behavior (Porway, 2014).

The concepts introduced in this section revolving around the three big issues of opacity, uncertainty, and emergence will become the basis for the articulation of the design principles of the module (see Section 4) and the analysis of the core activity of the module, where the different approaches to AI are introduced and compared by applying them on the simple case of tic-tac-toe (see Section 6).

4 Design Principles of the Module

The design principles on which the module on AI is constructed lie on the Model of Educational Reconstruction (Duit et al., 2012) and its epistemological orientation. This model concerns the transposition of domain-specific knowledge into knowledge that can be taught by instruction. The process of transposition, as the authors argue, is not only a matter of elementarization but of contextualization of the knowledge to make it fully accessible to

learners. In this way, the core scientific contents are enriched by perspectives that come from disciplines like history and philosophy of science, cognitive sciences, pedagogy, psychology, linguistics, or even from other disciplines when educational reconstruction is addressed with an interdisciplinary perspective. Hence, the result is not a simpler version of authentic disciplinary contents, but a more complex and stratified entity. We discuss now the design principles that guide the educational reconstruction of the contents related to AI. In particular, for the purposes of this paper, we focus on epistemic insights without providing many details on the other principles.

4.1 Design Principle #1: To Show the Connections Between AI and Society

In the module, time was devoted to making students explore the implications of the pervasiveness of AI applications along different dimensions (e.g., scientific, political, social, economic, ethical, and environmental), in order to make them understand that AI opens up nowadays professional opportunities and perspectives. In this context, specific attention was devoted also to making students explore the complex intertwining of humans and machines in the age of AI.

4.2 Design Principle #2: To Introduce a Variety of Approaches to AI and Programming Paradigms

In the module, within the large variety of methods and approaches, three programming paradigms are introduced as three possible ways in which an AI algorithm can be exploited: imperative procedural, declarative logic, and deep learning. The exposure to different approaches allows a recognition of the commonalities and specificities of each approach.

4.3 Design Principle #3: To Stimulate Reflections on Learning and Creativity Based on the Ways Machines Learn

Through exposure to different types of programming paradigms in AI, the module aims at making students reflect on their own learning (e.g., learning by following a set of instructions as in the imperative procedural paradigm, learning by moving within known constraints and sets of rules as in the declarative logic paradigm, or learning by being exposed to examples as in the ML paradigm). In this way, students are encouraged to develop their metacognition skills. Moreover, in the module, specific occasions are created to open up spaces to reflect on the meaning of human and artificial creativity.

4.4 Design Principle #4: To Keep the Technicality Level as Little as Possible

The module is not thought of as a learn-to-code experience but it aims at making students learn about AI by focusing on the main ideas behind the revolution that has been characterizing the last few decades. Hence, even if specific examples are presented and discussed in depth in class, the students were guided through the conceptual structure of the algorithms, without entering any technical detail of the programming languages. As we will detail in Section 5, implementations of the AI algorithms in Python (for the imperative-procedural paradigm) and Prolog (for the declarative-logic one) programs were given to students. These languages have a very light syntax and were always presented by instructors with

Table 1 Questions to foster epistemic insights on each epistemologically relevant issue identified in the literature

Issues	Questions to foster epistemic insights
Opacity	Is the method *interpretable* or inherently *opaque*?
	Is the approach *symbolic* or *sub-symbolic*?
Uncertainty	Has the output of the method a *deterministic* or a *probabilistic* character?
Emergence	In the approach, does knowledge flow *top-down* or *bottom-up*?
	Are *instructions*, *rules*, or *examples* given as inputs to the method?
	Does the method *imitate* or *generate* knowledge?

high-level descriptions. The implementation of a neural network (for deep learning) in Matlab was never shown to students, as they interacted only with a graphical interface.

4.5 Design Principle #5: To Exploit Epistemic Insights as a Way to Compare Different Approaches to AI

As shown in Section 3, AI touches many epistemologically relevant issues. Our choice is to introduce them in our module on AI to foster the comparison between imperative procedural, declarative logic, and deep learning approaches, in the form of epistemic insights (Billingsley, 2017), which we conceptualize here as questions to make students reflect on the knowledge taught. We are not referring here to "big questions" (Billingsley et al., 2018)—that, however, could have been pertinent—like "what is human intelligence compared to artificial intelligence?", "what is knowledge?", and "what is the destiny of the relationship between humans and machines?". In particular, the epistemic insights that we foster in the module to compare the three programming paradigms are more fine-grained and can be grouped into three blocks, each corresponding to one of the epistemologically relevant issues outlined in the framework (opacity, uncertainty, and emergence). In Table 1, we report, for each issue, the questions through which each programming paradigm was analyzed in the module.

4.6 Design Principle #6: To Connect Epistemic Insights to Operational Vocabulary

The epistemic insights are explicitly addressed in the module and introduced to students as a way to compare the approaches. However, they are linked to an "operational vocabulary" that consists of terms that, on one side, are directly recognizable in the conceptual layer of teaching the three different approaches and, at the same time, can foster the comparison of the dimensions identified by the epistemic insights. This resonates with what was pointed out by Billingsley and Ramos Arias (2017): "by including the term insight and by referring to knowledge about knowledge, we seek to signpost that a strategy to promote epistemic insight is not the same as a course to teach epistemology. [...] Adding a focus on discovering and advancing students' epistemic insight in schools encourages teachers to find pragmatic approaches to helping students make better sense of the messages they receive in different subjects about the nature of knowledge across the subject boundaries" (Billingsley et al., 2018, p. 1121). In our module, examples of terms that allowed the epistemic insights to be anchored to the disciplinary realms are algorithm, the role of the

programmer, problem-solving, and learning. We argue about the role of these terms and how they connect to epistemic insights in the Discussion.

5 Overview of the Module

The module on AI has been developed within the I SEE project and successively refined within the FEDORA project. Like all modules elaborated within the I SEE project, it is articulated in five teaching-learning phases. A detailed description of the module is available in more extended works (Barelli, 2022; Ravaioli, 2020), and all resources (e.g., slides and worksheets) are available on the I SEE Project website (https://iseeproject.eu/i-see-module-on-artificial-intelligence/). In the following, we provide an overview of the activities that are most relevant for our reflection on epistemological issues and that will allow setting the context for the analysis of the activities on tic-tac-toe.

The module begins with students encountering the focal issue of the module. This phase is aimed at making students develop a preliminary level of awareness of the ways in which conceptual and epistemological scientific knowledge, the specific language, and the methodological and pedagogical approaches will interweave in the module. In the module on AI, this phase consists of two activities. In the first, two overview lectures are held: one by an expert in AI and the other by an expert in the science of complex systems. The lectures aim to introduce conceptual and epistemological knowledge that will be developed and examined in depth in the module. The focus is on (i) the development of AI over the last few years, (ii) the different approaches to teaching a machine "to reason" and to solve a problem, and (iii) the significance of studying a problem from the point of view of complexity. The second activity of the first phase of the module engages the students in constructing collaboratively a map of the state of the art of AI, as an overall picture of where AI can be encountered nowadays. Particular emphasis is placed on the different fields of AI applications (archeology, art, services, scientific research, …), the risks and potentialities of AI applications, and future changes in the job market and STEM careers that the use of AI has been leading to.

The second phase of the module contains the fundamental elements of the topic that students engage with. In this phase, conceptual knowledge (focused on the disciplinary contents which are reconstructed (Duit et al., 2012) from an educational perspective), epistemological knowledge and practice (such as the practices of modeling phenomena, arguing, and explaining), and inquiry practice (such as the skills of posing questions, formulating hypotheses, recognizing modeling as a process of isolating a particular phenomenon, and moving from models to experiments and vice versa) are deeply intertwined. This phase consists of three activities to introduce the three aforementioned approaches to AI: imperative procedural, declarative logic, and deep learning through neural networks. The lectures were interactive and carried out by the authors of the paper (a computer scientist, a mathematician, and two physicists) and included a first part with an introduction to the specific approach and a second one in which the approach was exploited to solve a common task: to make a computer play tic-tac-toe. An extensive description of how the paradigms were introduced to students and how the problem of tic-tac-toe was addressed is the object of Section 6.

The third phase of the module concerns the so-called "bridge" activities. They are crucial activities in the module since they need to connect scientific, conceptual, and epistemological knowledge and practice (which characterize the first two phases of the module) with the concepts of complexity that are necessary to address the issue of opacity,

uncertainty, and emergence from an epistemological and scientifically-grounded point of view. In the first activity of this phase, the distinctive traits of the three programming approaches to AI that are presented to the students in the second phase are compared in terms of the role that the programmer has in each of them, the strategy of problem-solving, and the meaning of algorithm, learning, and predictability. The second activity, instead, is connected with the second introductory lecture and connects the revolution of ML with the paradigm shift of complex systems: The features of complex systems are exemplified using simulations of Schelling's model of racial segregation (for emergent properties and non-linearity), the predator-prey Lotka-Volterra model (for non-linearity, feedback, and causal circularity), and Lorenz's model for meteorological predictions (non-linearity and deterministic chaos). During the lesson, it is pointed out that this new non-deterministic way of thinking about the future has also inspired branches of social sciences.

6 The Tic-Tac-Toe: Analysis in Imperative Procedural, Declarative Logic and ML Neural Network Paradigms

In this section, we analyze how the tic-tac-toe was exploited in the module as a way to switch on an epistemological reflection on the disciplinary and conceptual contents through epistemic insights. Hence, we focus on the second phase of the module, which is aimed at addressing and deepening the conceptual and epistemological knowledge introduced in the first part. Three main programming paradigms are introduced: imperative procedural, declarative logic, and deep learning. To do this, the problem of coding a tic-tac-toe player is addressed with each paradigm. Each paradigm was exploited in a programming language (Python, Prolog, or Matlab) to give the sense of how the reasoning is turned into a code. A special attention was paid to compare the high-level description of procedural reasoning, exploited in Python, against the structure of logical reasoning explored in Prolog.

To analyze the activity, we split each subsection into three paragraphs: the first is devoted to introducing the basic ideas at the basis of the specific programming paradigm, the second presents the implementation of the game of tic-tac-toe within that paradigm, and the third highlights the epistemic insights that the specific paradigm allows to stress. In both the first and second paragraphs, we make use of some footnotes to specify some details in a more formal way: however, they are not essential for reading and fully understanding the paper.

6.1 Imperative Procedural Paradigm

6.1.1 Programs as Implementations of Step-by-Step Algorithms

In the module, the students were led to get acquainted that, given an algorithm, meant as a finite sequence of unambiguous steps to solve a problem, it is possible to translate it into a program (written in an imperative procedural programming language) that will eventually (through layers of translation/interpretation) be executed on an electronic computer. In the imperative procedural paradigm, the machine is clearly told what actions—often organized in procedures—to execute. A program can be seen as the modification of the machine's internal state (i.e., the association between modifiable variable names and the associated value) during the execution.

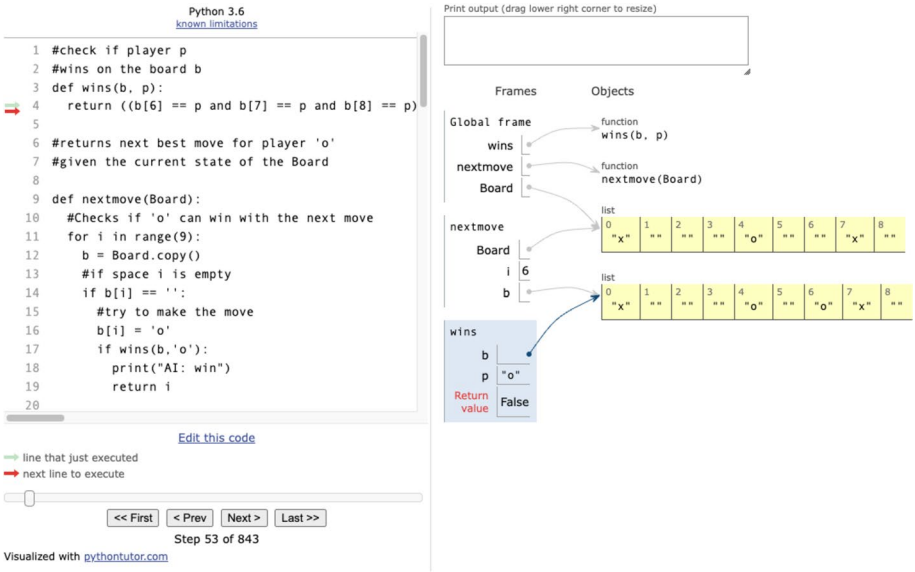

Fig. 1 A portion of a Python program that determines the next move in tic-tac-toe, together with a simplified representation of the state of the memory of the machine at a precise step of execution

Different high-level languages, like the most famous C, Java, and Python, were mentioned as examples of languages that could be used for imperative programming. In these kinds of languages, the execution steps can be precisely traced, and the memory state of the machine can be explored at each step (see Fig. 1).

6.1.2 A "Brute Force" Algorithm to Perfectly Play Tic-Tac-Toe

Tic-tac-toe is a simple game in terms of possible combinations: there are only 765 legal positions (by taking into account symmetries—rotations and reflections) on the board. Hence, it is very easy to find an algorithm that leads a player to play the *perfect game*, by always performing the moves that lead to the best possible outcome.

For example, Newell and Simon (1972) described eight simple rules the player should follow, by picking the first that is allowed by the current configuration, when determining the next move. We report them as summarized in ("Tic-tac-toe", 2023):

1. Win: If the player has two in a row, they can place a third to get three in a row
2. Block: If the opponent has two in a row, the player must play the third themselves to block the opponent
3. Fork: Cause a scenario where the player has two ways to win (two non-blocked lines of 2)
4. Blocking an opponent's fork: If there is only one possible fork for the opponent, the player should block it. Otherwise, the player should block all forks in any way that simultaneously allows them to make two in a row. Otherwise, the player should make a two in a row to force the opponent into defending, as long as it does not result in them producing a fork. For example, if "X" has two opposite corners and "O" has the center,

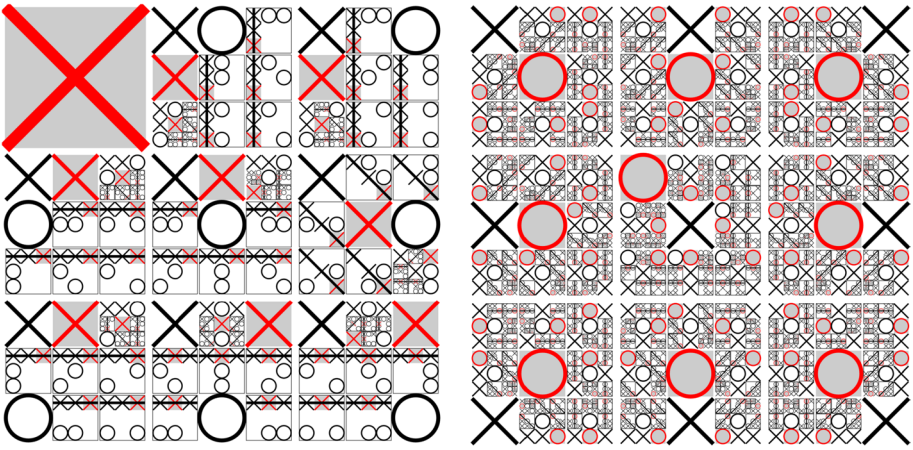

Fig. 2 In red, the optimal strategies for player X (left) and player O (right). In the right picture, for example, the big red Os are the ideal moves that the O player, which starts second, should do relative to what the X player has done as its first move (the big black Xs). The smaller pictures analogously indicate the strategy for the following moves. The figures are by nneonneo - Own works, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=12305931 and https://commons.wikimedia.org/w/index.php?curid=12305920

"O" must not play a corner move to win. (Playing a corner move in this scenario produces a fork for "X" to win)

5. Center: A player marks the center. (If it is the first move of the game, playing a corner move gives the second player more opportunities to make a mistake and may therefore be the better choice; however, it makes no difference between perfect players)
6. Opposite corner: If the opponent is in the corner, the player plays the opposite corner
7. Empty corner: The player plays in a corner square
8. Empty side: The player plays in a middle square on any of the four sides

While these rules are expressed in a human-understandable form, talking about "corners" or "forks," they are directly derived from analyzing all the possible games (see Fig. 2). Therefore, this can be called a *brute-force* algorithm: for each step, one simply considers all the possible evolutions (in other words, the game tree in Fig. 3) of the game, considering both player's and opponent's possible future moves, and executes the move with the best possible outcome. From an AI perspective, it is far from being an "intelligent" strategy. Indeed, the brute-force strategy is applicable because of the few possible states and games that tic-tac-toe presents. With games like Chess or Go, where the possible states are in the order of 10^{47} and 10^{170}, respectively, a brute-force approach would take an amount of time way larger than the age of the universe, making it infeasible.

We implemented the tic-tac-toe rules in a straightforward Python program that, given the current configuration of a tic-tac-toe board, perfectly[1] determines the next move. We report it in Annex 1.

[1] Due to time and space limitations, the actual program given to students is only "almost" perfect, but our simplification does not hinder the objectives of the activity.

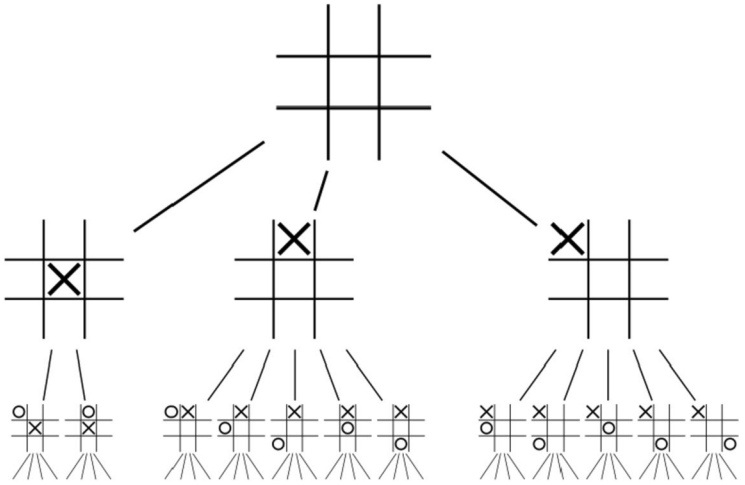

Fig. 3 Part of the game tree for tic-tac-toe, i.e., the tree that follows all possible moves of the game

6.1.3 Epistemic Insights of the Tic-Tac-Toe Within the Procedural Paradigm

In this approach, top-down knowledge, in the form of an explicit play algorithm and a representation of the problem's data, is given to the machine for it to follow and play, i.e., to know the specific actions to perform (emergence: top-down flow of knowledge). Therefore, the programmer needs a deep understanding of the problem at stake, to choose an adequate representation for the data (for example, a list/matrix of 9 elements that are "X," "O," or "blank"), to devise an ideally optimal strategy to solve the problem for every possible situation, e.g., by analyzing what happens in different configurations of the game (see Fig. 2), to describe it as an algorithm and implement it in a program (emergence: instructions as input).

Since explicit, procedural knowledge on what to do is given to the machine, it is always possible for a human to read (opacity: symbolic approach), at a chosen level of abstraction (high-level program, machine code, and even the logic gates or the transistors), and associate a meaning to each step performed by the machine (opacity: interpretable method).

This gives programmers total control over the results: the output is deterministic (same input and same output), and, when tested and debugged, or even formally verified, the correctness of the solution is guaranteed (uncertainty: deterministic output).

The machine's predictability, together with the fact that it executes what it is instructed to do and in a specific order, does not allow it to show any form of creativity, at least if you are aware that it is following a known, pre-determined sequence of steps (emergence: imitation of knowledge). Of course, the creative process already occurred in the programmer's work, when he devised and implemented the algorithm.

6.2 Declarative Logic Paradigm

6.2.1 Computation as a Logical Deduction

The declarative logic paradigm is based on logical reasoning[2]. The programmer specifies a set of true *facts* and *rules* (logical implications) as the *knowledge base*[3] of the program. The user of a logic program can ask for a *goal* (a conclusion they are trying to prove true), and an *inference engine* (a unique algorithm, common to all logic programs, that implements an inference procedure, based on *modus ponens*[4]) tries to apply facts and rules to derive the *goal*. We can think at the inference engine exploring a search tree that represents all possible choices[5] for facts and rules to apply in order to prove the goal, backtracking, and trying different choices when finding a "dead end." The necessary ability of the programmer is, in this case, to formulate the problem in terms of true propositions and logical inferences based on the truth of propositions and material implications.

We provide a simple example of a problem that can be formulated in a logical way. We start from a set of facts ("Giovanni is Anna's father," "Carlo is Antonio's father," "Andrea is Carlo's father," "Andrea is Giovanni's father") and rules (material implications that are considered to be true, e.g. "If it is true that X is the father of Y and Y is the father of Z, then X is the grandfather of Z"). At this point, we can create a logic program that logically deduces if a proposition ("Andrea is Anna's grandfather") is true or not. To implement this example in a programming language, we choose Prolog, the most widely used one. In Prolog, it is expressed with the following program (you can read ":-" as logical implication "←", i.e., to be read from right to left; you can read the comma as a logical "and"):

`father('Giovanni', 'Anna').`	Giovanni is Anna's father
`father('Carlo', 'Antonio').`	Carlo is Antonio's father
`father('Andrea', 'Carlo').`	Andrea is Carlo's father
`father('Andrea', 'Giovanni').`	Andrea is Giovanni's father
`grandfather(X, Z) :- father(X,Y), father(Y,Z).`	X being Y's father and Y being Z's father imply X being Z's grandfather

Hence, to answer the question:

`?- grandfather('Andrea', 'Anna').`	is Andrea Anna's grandfather?

[2] Formally, it is based on first-order predicate calculus.

[3] Using the so called *closed world assumption*, it describes everything that is true.

[4] More formally, it uses the *SLD resolution* (Robinson, 1965) inference rule and a *unification* algorithm to assign values to free variables.

[5] Note that, in principle, the paradigm maintains a form of non-determinism: the result of a computation is a set of computed answers, deriving from all possible choices that lead to success. Of course, when implementing this paradigm in an actual programming language, that runs on a deterministic machine, a choice should be made. For example, various implementations of Prolog choose to use the textual order in which the facts and rules appear in the program, giving the opportunity to the programmer to ask for more solutions if they will. For a more formal, CS-oriented discussion on the logic paradigm, see (Gabbrielli & Martini, 2010, ch. 12).

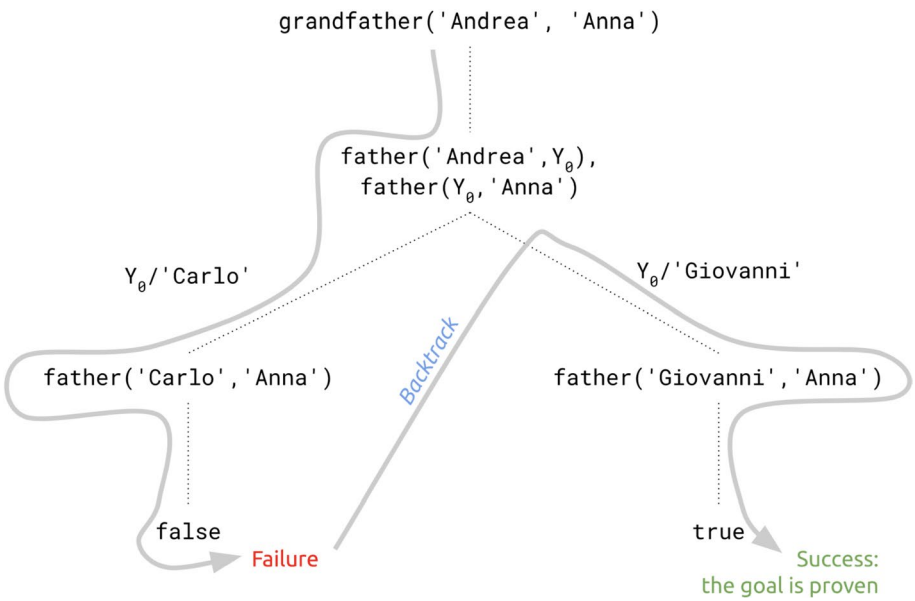

Fig. 4 Search tree for the goal "?- grandfather('Andrea', 'Anna')". A possible execution flow is depicted with the gray line and the colored labels

The program generates (implicitly) the search tree in Fig. 4, also called proof tree, to arrive at the proof of the statement.

6.2.2 Tic-Tac-Toe in Prolog

To program a tic-tac-toe player in Prolog, the knowledge about the game and the strategy has to be turned into a set of facts and rules and the moves can be obtained by exploring a search tree with a logical deductive approach. In particular, the students were guided to recognize that, in the case of tic-tac-toe, the knowledge base includes facts like winning combinations, rules like inferential statements about what it means to win, or to avoid the other player winning, and to create or to avoid a configuration where it is sure that one wins in the next move (a fork), and derived true statements are moves. We report an example of knowledge base representation, adapted from Scott (2009, p. 563) that was shown to the students to visualize how the reasoning is turned into a code. A possible winning configuration corresponds to having the same symbol (e.g., "x" in the last line (starting labeling the cells from 1, from left to right, and from top to down, corresponding to having "x" in positions 7, 8, and 9). In the code, it can be expressed by the fact:

```
win_line(7,8,9).
```

To consider all the equivalent configurations and say they are winning ones ("tic-tac-toe"), it is necessary to include:

```
tictactoe(A,B,C) :- win_line(A,B,C).
tictactoe(A,B,C) :- win_line(A,C,B).
tictactoe(A,B,C) :- win_line(B,A,C).
tictactoe(A,B,C) :- win_line(B,C,A).
tictactoe(A,B,C) :- win_line(C,A,B).
tictactoe(A,B,C) :- win_line(C,B,A).
```

Here are two examples of rules expressed in the form of material implications, where the symbol % indicates comments, ignored by the Prolog machine:

```
                    % I win if I complete a line
% (I already have 'o' in B and C, so I put the third 'o' in A)
   win(A) :- o(B), o(C), different(B,C), tictactoe(A,B,C).

                % I prevent 'x' victory by choosing A
    prevent_other_win(A) :- x(B), x(C), different(B,C),
                            tictactoe(A,B,C).
```

By asking the Prolog program for the goal:

```
next_move(A), empty(A), assert(o(A)).
```

The user can expect it to produce an assignment of a number from 1 to 9 to the variable A that corresponds to the computer putting an “o” in that cell.

We remark, by looking at the full code reported in Annex 2, that the programmer has exploited the fact that Prolog considers facts and rules in textual order (and therefore the execution results are deterministic and predictable) to enforce a strategy, by putting the preferred rules (like win(A) :- …) first.

6.2.3 Epistemic Insights of the Tic-Tac-Toe Within the Declarative Logic Paradigm

In the declarative logic approach, the machine’s “brain” does not have any instruction in advance of the moves to perform step by step to play the game. However, knowledge about the rules of the game and the inferential mechanism that produce information about the next move starting from the previous ones are incorporated from the very beginning in the code. Its capacity to make a move on the basis of a configuration of the board is obtained top-down, starting from the facts (true premises) and rules that are provided (emergence: top-down flow of knowledge). The declarative logic program used to play tic-tac-toe allows focusing on the type of data to put as an input and the kind of knowledge produced as an output. Winning rules and information about the moves expressed in the form of true

premises have to be formulated so that the machine performs a deductive *modus ponens* reasoning, producing new necessarily true statements (new moves to perform) derived from true statements included as inputs. The knowledge infused in the machine is not only possible moves but also rules, through which the machine can produce new ones (emergence: rules as input).

An implication of the top-down character based on rules of the declarative logic approach is that the programmer clearly expresses the rules of the game and the tree of possible moves is implicit in its decisions, and the results of its elaboration of premises through rules can be visualized and checked. There can be human errors in the writing of rules or in the premises, thus deriving as true propositions that we other humans can consider false, but the choices and all the possible conclusions are explicit and can be checked and modified. A human can make sense of the code and change it to obtain different and predictable outcomes (moves), because a symbolic relationship exists between the knowledge in input and output, thus between the state of the process and the decision obtained by the machine (in our case the next move in the match) (opacity: symbolic approach). This allows the users to actually decide the strategy of the machine in solving the problem since the rules are explicit and the relationship input-output is human-readable (opacity: interpretable method).

Once set up the starting knowledge and the rules, the solution tree is unique. This reveals an epistemic characteristic of the knowledge produced by Prolog: new knowledge is produced, but it is totally predictable. No creativity is possible for the machine (emergence: imitation of knowledge). Another epistemic insight that we can highlight through the tic-tac-toe is that the Prolog machine produces completely accurate and predictable solutions; thus, its behavior is in the domain of certainty and can be determined *a priori* (uncertainty: deterministic output).

6.3 Neural Network Paradigm

To analyze how the epistemic insights were integrated into the discussion of the tic-tac-toe problem through an ML approach, a brief conceptual introduction is needed on the specific family of algorithms we used in the module, i.e., deep learning with feed-forward artificial neural networks.

6.3.1 Neural Networks 101

To describe what a neural network is, we have to consider that this family of algorithms has been developed with the aim of imitating the functioning of neurons and neuronal networks. A neuron can be modeled as a functional unit that (i) receives via the dendrites (input wires) signals that come from other neurons or from the external world, (ii) processes the signals, and (iii) sends the signals processed to another neuron via the axon (output wire). Starting from the biological model of a neuron, in 1958, Rosenblatt modeled the first, simplest artificial neural network with a single neuron as a computational unit that (i) receives a certain number of inputs, (ii) carries out calculations, and (iii) returns the results of the calculations (Rosenblatt, 1958).

A neural network consists of many single neurons organized in layers, according to a variety of possible arrangements and resulting architectures. In the following, we focus on the feed-forward neural network, characterized by the fact that the connections

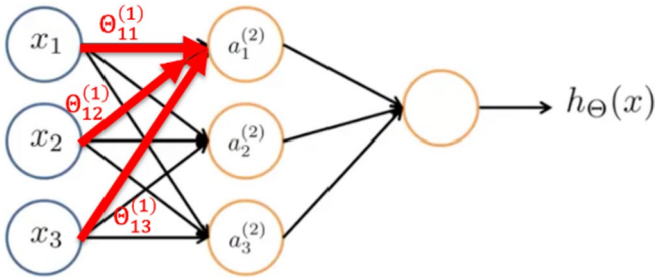

Fig. 5 Schema of a feed-forward neural network with three features as inputs and three neurons in the hidden layer

between the nodes cannot form a cycle and that information only flows from the input to the output. A graphical example is reported in Fig. 5. In this network:

- Each connection from an element of one layer to one element of the next has a weight $\theta_{ij}^{(k)}$, where i is the index of the neuron in the destination layer, j is the feature's index in the starting layer, and k is the index of the starting layer
- Each neuron computes its activation function a by applying the sigmoid function g on the scalar product between the values of the inputs and the corresponding weights. For example, the activation function computed by the first neuron in the second layer is $a_1^{(2)} = g\left(\theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3\right)$
- The last neuron provides the final computation of the hypothesis function, calculating the sigmoid on the scalar product between the results of the activation and the corresponding weights: $h_\theta(x) = g\left(\theta_{11}^{(2)}a_1^{(2)} + \theta_{12}^{(2)}a_2^{(2)} + \theta_{13}^{(2)}a_3^{(2)}\right)$

The neural network is named feed-forward because the process starts with the inputs that pass to the activations of the hidden layer which forward propagate the results to compute the activations of the output layer. Moreover, we can notice the complexity of the resulting hypothesis function: what is fed into the final formula are not the features x that describe the input data, but the values $a_i^{(j)}$ which are in turn the results of the computations performed by the hidden layer which are learned as functions of the input. Therefore, $h_\theta(x)$ results a complex function, strongly non-linear.

Once described the mathematical form of the functions, the issue is to choose the values of the weights which characterize the connections between the nodes of the network. To assess the appropriate values of the weights, the network is trained through a process named error back-propagation which follows a few steps:

1) In the initial iteration, the weights are initialized randomly
2) The process goes on by utilizing the features x of the first example within the training dataset. These features are employed to drive forward propagation through activation functions in the hidden layer, followed by computation of the hypothesis function of the final layer

3) The error is computed by comparing the known label associated with the training example[6] to the prediction obtained from the hypothesis function
4) This error is then "back-propagated" from the output layer to the hidden layer, which involves adjusting the weights to minimize the cost function when recalculated with these updated weights

Steps 2, 3, and 4 are repeated for all the examples in the training dataset.

After the training phase, the neural network is tested on new examples to evaluate the accuracy of the model, which is computed as the fraction of the test examples for which the prediction corresponds to the label with respect to the total number of tests. This accuracy becomes a measure of the network's reliability to generalize the classification to new data.

6.3.2 Neural Networks for Tic-Tac-Toe

To play tic-tac-toe with an ML approach, and in particular with a neural network, we imagine a robot named Lucy who has to learn how to play this game effectively and the issue for the programmers consists in "building a brain" that makes her capable of performing the task. The steps to build her brain are (i) to build a database from which she can learn, (ii) to choose the type and the structure of the network, (iii) to train the network, and (iv) to test Lucy's ability in playing tic-tac-toe. The result of this process will be the network's decision of which move to do (output) when provided with a specific situation of the grid (input). These steps were the same that the instructor presented to the students, interacting with the Matlab interface to create the games for training and to test the networks' ability to play. The Matlab code behind the interface was not inspected nor shown to the students, because the focus was never on making students understand how to code a neural network but to grasp the overall meaning of training and testing a ML algorithm.

Construction of the Database To build the database, the 3×3 grid of the tic-tac-toe is mapped into a 9-component vector as in Fig. 6.

Then, we need to develop a sense of how matches and moves can become a database. Before the first moves, the grid is empty and the vector has all its components set to 0. Then, player 1 draws "1" in a cell and the corresponding component of the vector takes 1 as a value. Then, player 2 does the same, and so on, until the game ends. Nine moves of one match are represented in Fig. 7.

The database is constructed by generating a number of matches, by making play:

- An imperative algorithm (which operates with the strategy described in Section 6.1.2) against a random player (which selects a random cell among the free ones), or
- Two random players against each other, or
- Two imperative players.

In Fig. 8, we report the Matlab interface through which the students were guided to construct the moves database. The Matlab code to produce this output was not inspected

[6] In the case of supervised learning, the database is such that to each example the true label is associated; for example in the case of a neural network built to distinguish dogs from wolves, in the database each image is flanked by the true label of "dog" or "wolf."

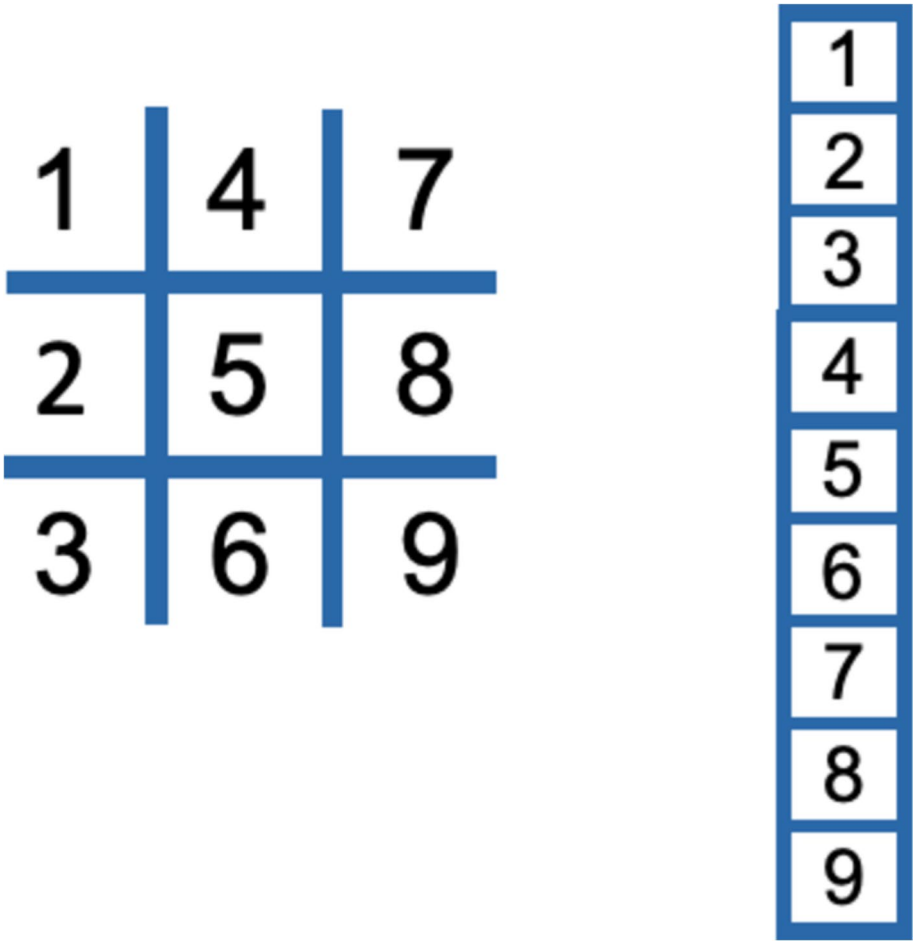

Fig. 6 Mapping of the tic-tac-toe grid into a vector

with the students since, for the scope of the course, we stressed the high-level structure of problem's setting.

In each match, the vectors of winner's moves (in the case of Fig. 7, the winner is player 1) are defined as the *target* moves given the *example* of the loser's moves. Considering the example of the game depicted in Fig. 7, the corresponding database is reported in Fig. 9.

Choice of the Network's Architecture The neural network to solve this problem is chosen with a feed-forward architecture. To define the number of inputs and the number of outputs, we must consider the characteristics of the specific problem: given any board (a 9-component vector), the task of the network is to choose the next move (another 9-component vector). Hence, the number of input elements must equal that of output elements, and both must equal 9. Regarding the other parameters of the network, it is chosen with a single hidden layer. The choice of the number of neurons in the hidden layer and the portion of the database to train the network is set through the interface in Fig. 10.

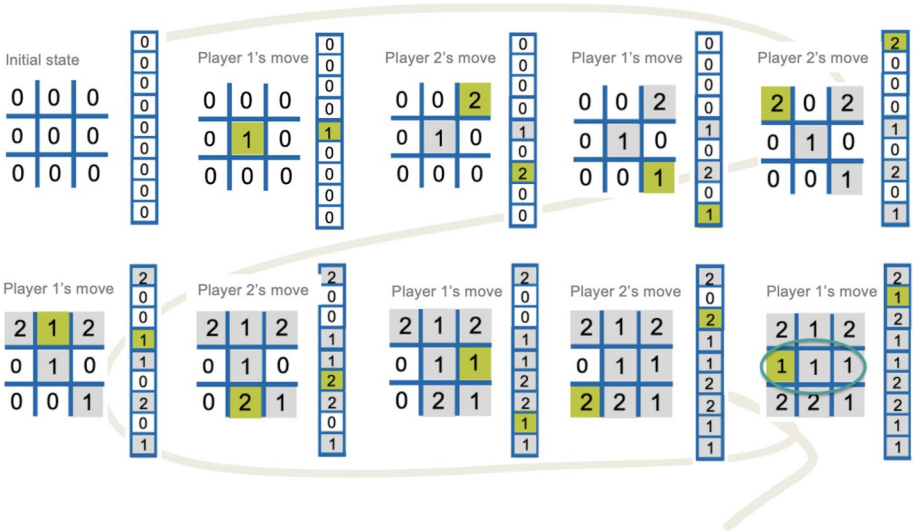

Fig. 7 Moves in a match between players 1 and 2, with both grid and vector representations

Network's Training and Test To teach the network how to play tic-tac-toe, part of the database is selected to train it. After the training phase, which includes the error back-propagation algorithm, the capacity of the network to win other matches is computed. To do that, the instructor guides the students to experiment in a virtual arena in which, through the Matlab interface as exemplified in Fig. 11, it is possible to select the players of the game (e.g., human, imperative, random, or neural network—and in this latter case which neural network) and monitor the number of games won by the players. Doing this, we can observe that the success rate of the network increases with the number of matches on which it is trained; even when trained only with a database made by the moves of two random players, the network succeeds in winning in most cases. On the opposite, when trained only with the matches played by two imperative algorithms, the network fails in case of a random player's moves.

6.3.3 Epistemic Insights of the Tic-Tac-Toe Within the Neural Network Paradigm

As a neural network, Lucy's brain did not have any top-down knowledge in advance of the most efficient strategy to play the game. On the opposite, its capacity to make a move on the basis of a configuration of the board is obtained bottom-up, starting from the examples that are provided (emergence: bottom-up flow of knowledge). The supervised learning algorithm used to play tic-tac-toe also allows focusing on the type of data that feeds the network: matches have to be created and moves have to be labeled as winning or losing so that the machine learns. Indeed, the knowledge infused in the machine is not only data but also a true label, through which the network can learn (emergence: examples as input).

A consequence of the bottom-up character of the artificial neural network is that, even if we say that it "learnt" to play tic-tac-toe, actually this learning consists in having assigned numbers to connections between neurons (i.e., the weights) through the complex process of error back-propagation. To a human, those numbers are meaningless because no symbolic relationship exists between the values of the parameters and the decision obtained by

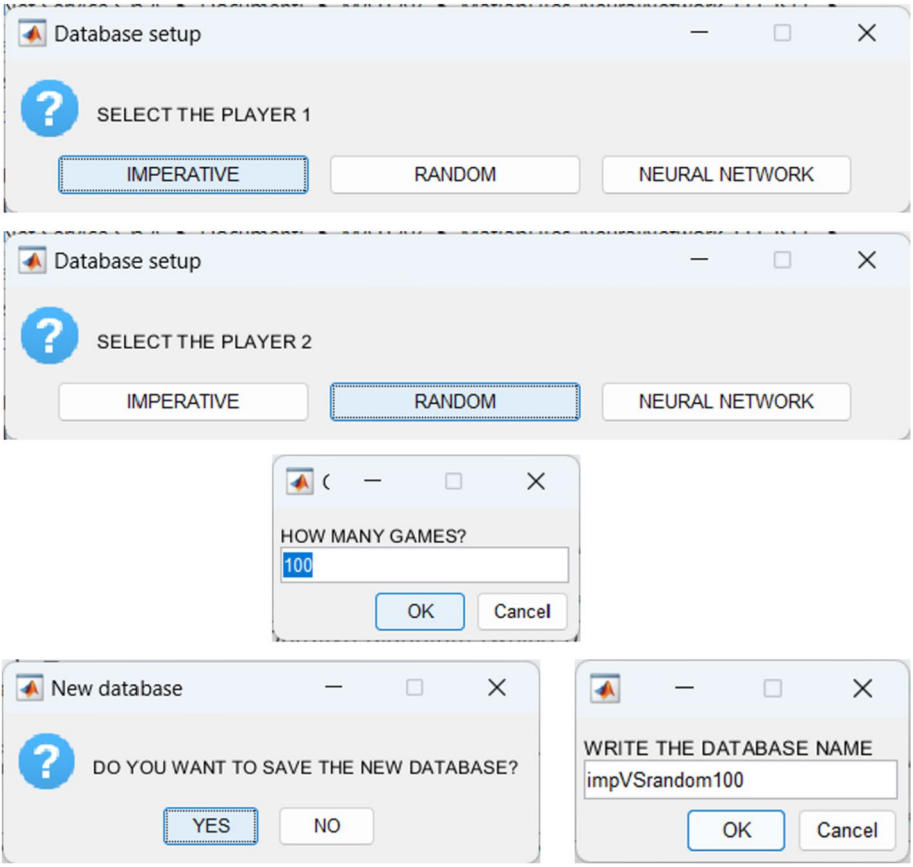

Fig. 8 Matlab interface to build the database by making an imperative algorithm (player 1) play against a random one (player 2) 100 times

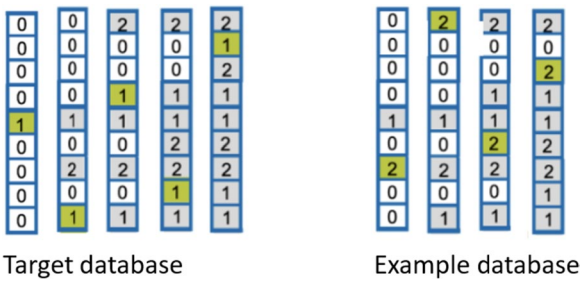

Fig. 9 Portion of target and example database related to the game depicted in Fig. 7. The target database is the collection of the vectors representing the moves of the winner. The example database is the collection of the vectors representing the moves of the loser

the machine (in our case the next move in the match) (opacity: sub-symbolic approach). This prevents the users from actually interpreting the strategy of the network in solving the problem: it is indeed a black box that obtains implicit knowledge through examples, without making explicit and human-readable why these results are obtained (opacity: opaque method).

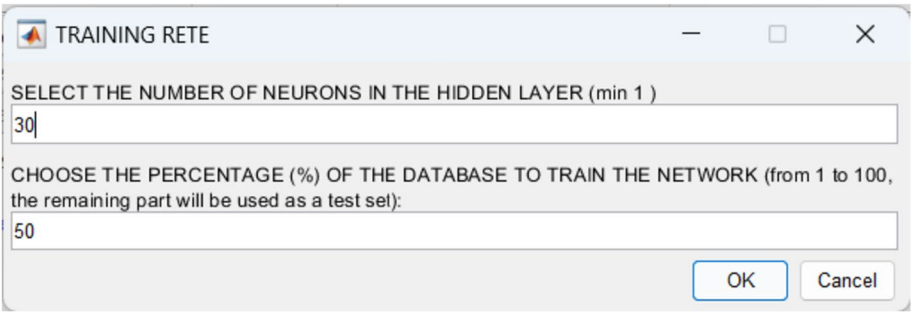

Fig. 10 Matlab interface to construct a 1-layer network with 30 neurons in the hidden layer. In the same window, the user can choose the portion of the database to use for training (default: 50%)

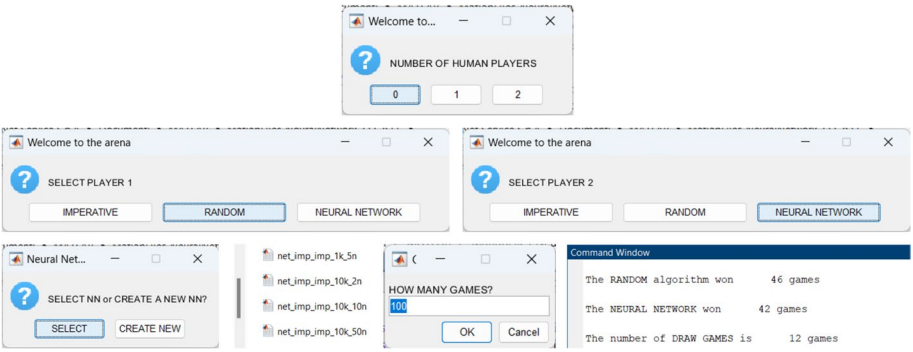

Fig. 11 Matlab interface to make a random player play 100 times against a neural network (in this specific case, the neural network chosen is trained on the matches played by two imperative algorithms) and results of the matches

Moreover, the different ways of training the neural network lead to unexpected behaviors like success when learning from random players and failure when learning from perfect imperative players. This reveals an important epistemic characteristic of the knowledge produced by the neural network: since it is not *a priori* infused in the form of an algorithm, we obtain something that resembles creativity, i.e., the original solution of a problem without a previously determined path to follow (emergence: generation of knowledge). One further epistemic insight that the tic-tac-toe allows to stress is that the network never comes alone: it is always flanked by its accuracy, determined in the testing phase. Differently than in the cases in which the strategy is given to the machine, the way in which the neural network works can only lead to a probability of success in performing the task, not in a certainly accurate solution (uncertainty: probabilistic output).

7 Discussion and Conclusions

In writing this paper, we were guided by this research question: *Which epistemic insights should and can be introduced to upper secondary-school students to make them aware of the AI and ML revolution?* In presenting the solution to the problem of tic-tac-toe

through the imperative procedural, the declarative logic, and the ML neural networks approaches, we have pointed out that different epistemic insights can be fostered. In particular, the epistemic insights have here the role of articulating the difference between the three approaches, hence highlighting the novelty of ML with respect to traditional approaches to programming. In Table 2, we summarize the difference in the approaches across the epistemic insights.

Regarding the issue of *opacity*, we have argued that imperative procedural and declarative logic paradigms are interpretable and symbolic, while the ML neural network one is opaque and sub-symbolic. With the first two approaches, the steps the machine follows are intelligible and can be meaningfully connected to "symbols" related, at a certain level of abstraction, to the issue at stake. In the case of the imperative procedural approach, every instruction and output of the procedure implemented can be checked in the code directly (each instruction written in the code corresponds to one of the symbolic steps for the solution of the problem). In the declarative logic approach, instead, what is expressed in a symbolic, visible, and evaluable way in the code are the logical rules and the true facts, and the relationship between input (knowledge base) and output (deduced knowledge) can be analyzed entirely checking the search tree explored by the inferential engine. The ML neural network paradigm is very different. The distributed structure of the neural network in many individual agents (the neurons) and the nonlinear character of interactions among them create an opaqueness of the network. At the end of the process of training, validation, and testing, the knowledge obtained by the machine that it uses to produce a result or a decision does not consist of human-readable information but only in a wide matrix of connections' weights expressed as numbers.

The difference between the imperative procedural and declarative logic paradigms, on one side, and the ML neural networks on the other, was made explicit in the analysis of the tic-tac-toe. With the first two approaches, we are able to associate at every stage of the computation a meaning to the actions performed by the machine and, as a consequence, is rather easy to explain the results obtained, whether they be correct or incorrect. This is not the case of the neural network where its structure cannot be interpreted as any logical decision or procedure, without explicit and symbolic correspondence with the issue at stake.

Second, regarding the issue of uncertainty, we have shown that imperative procedural and declarative logic approaches are deterministic, while the ML neural network one is probabilistic. Within the first two approaches, given a particular input, the same output is always obtained, with the underlying machine always passing through the same sequence of states. This gives programmers total control over the results: being the output deterministic after programs are tested and debugged (or even formally verified), its correctness is guaranteed. On the opposite, the ML neural network paradigm is intrinsically probabilistic: to train the network, the weights are randomly initialized, and the nonlinearity of the functions leads to results that strongly depend on initial conditions, on the network's structure, and on the examples provided. At the end of the training, the neural network produces the same outcome if the same input is submitted; however, considering how the neural network is trained, from a statistical point of view, we can only talk about the efficiency of the network and the accuracy of the results. These epistemological aspects were shown through the example of the tic-tac-toe, making students reflect on the fact that in the first two approaches, it could have been easily verified that given each possible configuration in input, the output of the Python or Prolog program would have been the perfect move. On the contrary, for the chosen neural network, results were given in terms of the percentage of wins, losses, and draws. Moreover, training the network with different datasets had

Table 2 Summary of the epistemic insights declined into the three approaches to AI introduced in the module

		Procedural imperative	Declarative logic	Deep learning
Opacity	Is the method *interpretable* or inherently *opaque*?	Since explicit, procedural knowledge of what to do is given to the machine, it is always possible for a human to read and **interpret** the meaning of each step performed by the machine.	Users can decide the strategy of the machine in solving the problem since the rules are explicit and the input-output relationship is **interpretable**.	The neural network is an **opaque** black box that generates implicit knowledge through examples, without making explicit and human-readable why these results are obtained.
	Is the approach *symbolic* or *subsymbolic*?	Each step performed by the machine can be interpreted in a **symbolic** way (at a chosen level of abstraction: high-level program, machine code, even the logic gates or transistors).	A human can make sense of the code and change it to obtain different and predictable outcomes, because a **symbolic** relationship exists between the knowledge in input and output, thus between the state of the process and the decision obtained by the machine.	Learning consists of the machine having assigned numbers to connections between neurons. To a human, those numbers are meaningless because **no symbolic** relationship exists between the values of the parameters and the decision obtained by the machine.
Uncertainty	Has the output of the method *deterministic* or *probabilistic* character?	The output is **deterministic** and, when tested and debugged, the correctness of the solution is guaranteed.	Completely accurate and predictable solutions are produced: the behavior is **deterministic**.	The way in which the neural network works can only lead to a **probability** of success in performing the task, not in a certainly accurate solution.

Table 2 (continued)

		Procedural imperative	Declarative logic	Deep learning
Emergence	In the approach, does knowledge flow *top-down* or *bottom-up*?	**Top-down** knowledge, in the form of an explicit play algorithm and a representation of the problem's data, is given to the machine for it to follow and play.	Knowledge about the rules and the inferential mechanism that produces information is incorporated from the very beginning in the code. The machine's capacity of performing the task is obtained **top-down**, starting from facts (true premises) and rules.	The neural network does not have any top-down knowledge in advance of the most efficient strategy to solve the task. This capacity is obtained **bottom-up**, starting from the examples that are provided.
	Are *instructions*, *rules*, or *examples* given as inputs to the method?	The strategy to solve the problem for every possible situation has to be given as input in the form of a set of **instructions.**	The input given to the machine consists of **true premises** and **rules** that have to be formulated so that the machine performs a deductive *modus ponens* reasoning.	The input from which the neural network learns is made by **examples** in the form of data and their labels.
	Does the method *imitate* or *generate* knowledge?	The machine's predictability, together with the fact that it **executes** what it is instructed to do and in a specific order, does not allow it to show any form of creativity (at least if you know that it is following a known, pre-determined sequence of steps).	Once set up the starting knowledge and the rules, the solution tree is unique. This reveals an epistemic characteristic of the knowledge produced: **new knowledge** is produced **but** it is totally **predictable**. No creativity is possible for the machine.	Knowledge is **generated** by the neural network and, since it is not *a priori* infused in the form of an algorithm, we often obtain something that resembles creativity, i.e. the original solution of a problem without a previously determined path to follow.

different results: for example, giving it only examples of perfect matches gives much worse results than giving it examples of matches between a perfect and a random player.

Finally, regarding the issue of emergence, we have argued that imperative procedural and declarative logic paradigms are top-down approaches, while the ML neural network one is bottom-up. With the first two approaches to computing, knowledge of the problem is owned by the programmers who formulate the algorithms and impose their strategies on the machine that has only to apply them in the established way. The main difference between the two approaches consists in the fact that in the imperative procedural approach, knowledge is expressed in terms of step-by-step instructions and all of them must be known and written by the programmer, while in the declarative logic paradigm, the machine produces knowledge by implementing the given rules through the inference engine. With the connectionist approach, we move a step forward: knowledge emerges bottom-up from the examples with which the network is fed, without any *a priori* expert knowledge about the task. Differently than the case of the declarative logic paradigm, no rule is given to the machine but all knowledge and "learning" obtained by it emerges from input data.

The application of the lens of epistemic insights to the case of tic-tac-toe shows the difference between top-down and bottom-up approaches in a clear manner. Using the neural network, the programmers do not write any procedure for the machine to act the moves (as in the imperative procedural paradigm) nor a problem-specific set of formal rules and constraints to satisfy (as in the logical declarative one): it is the machine that trains itself "observing" winning and losing matches and, from here, learns its strategy. The strategy learned by the network can be very different from the optimal one because it is highly sensitive to the training dataset, and this can result in "creative" strategies. Moreover, the tic-tac-toe allows us to clarify a distinction between the two top-down approaches. In the imperative procedural case, the programmer needs to specify every possible move and describe them in an algorithm. With the declarative logic approach, the programmer does not put all the knowledge about every move in the code, because the inference engine allows the machine to elaborate information and produce new ones (deduct) in terms of concrete moves to make in a configuration: the machine is not creative in terms of strategies but is somehow autonomous in deciding how to act at every single step respecting the rules. As a consequence, we can say that the knowledge obtained as output in the three paradigms is respectively executed knowledge (imperative procedural), deduced knowledge (declarative logic), and generated knowledge (ML neural networks).

A final comment is needed on the way in which the epistemic insights were addressed in the module and introduced to students to compare the approaches. The concepts of opacity as opposed to transparency, uncertainty as opposed to determinism, and emergence as opposed to reductionism were explicitly mentioned to the students. However, the focus was never on epistemology *per se*. According to Design Principle #6, the epistemic insights that we have discussed in this paper were anchored to an operational vocabulary made of terms chosen to be, at the same time, closer to students' experience in school disciplines and that had the potential to trigger a reflection on the epistemic insights. In the following, we explore three terms of this operational vocabulary on which the comparison between the approaches was established.

The first is the concept of *algorithm*. This had been encountered by students, when learning computer science at school, as a step-by-step method to solve a problem: like a recipe, a sequence of steps, to obtain a result. In the module, it was stressed that this idea of conceiving the algorithm for the solution of a problem pertains to the imperative procedural paradigm only. Indeed, in a neural network, it is impossible to establish a correspondence between the actual steps of the underlying algorithm and

the symbolic meaning of the procedures to solve a task. Even in the declarative logic approach, the algorithm is different than in the imperative procedural: in this approach, the algorithm is the inference engine, and it is common to all logic programs, independently of the specific problem to be solved. Treated in this way, the term "algorithm" became leverage to introduce a reflection on the epistemic insights connected to *symbolism* and *interpretability*.

The second operational expression is the *role of the programmer*. Indeed, each approach allowed to highlight a different level and kind of knowledge that has to be owned by those who implement an AI algorithm. In the imperative procedural, the programmer needs a deep understanding of the problem at stake, to choose an adequate representation of the data, and to devise a strategy to solve the problem for every possible situation. In the declarative logic paradigm, programmers know in advance the variables of the problem, their constraints, and how to express what it means to have solved the problem: they only need to turn possible winning moves into facts expressed in the form of true premises (i.e., propositions in first-order logic with no free variables, choosing only the necessary information) and to transform the rules of the game in declarative statements of the kind "A implies B" where propositions are parts of the facts, in the suitable programming language. Finally, in the ML neural network approach, a programmer does not need to know in advance how the problem can be solved, but they need to model the problem in a way that can be fed to the neural network, to decide the architecture of the neural network itself, and to provide the adequate database of examples. Grounding the discussion on the role of the (expert) human in setting up the AI algorithm, it was possible to introduce the epistemic insights connected to *emergence*, especially to the relationship between inputs and outputs.

The final concept that we chose to analyze here and that was deeply addressed in the module, also in the light of Design Principle #3 ("To stimulate reflections on learning and creativity based on the ways machines learn") is that of *learning*. The imperative procedural paradigm conveys an idea of learning as a linear, deterministic process in which passing through a set of steps it is possible to achieve a result. In the declarative logic one, learning is instead the result of a process of proof that, from true premises and through the inference engine, allows to arrive at a conclusion; in the way that the declarative logic paradigm was introduced in the module (i.e., with the PROLOG language), the result of the application of the inference engine on the same facts and premises is deterministic, as in the previous approach. In the ML neural network paradigm, instead, learning does not include any execution or proof: knowledge emerges from examples and trials and errors and is intrinsically probabilistic. In this way, reflecting on learning was the way through which the epistemic insights connected to *uncertainty* and *emergence* were introduced.

Further analysis needs to be conducted on the relationship between the operational vocabulary and the STEM disciplines at stake, on one side, and the epistemic insights, on the other (Ravaioli, 2020). Indeed, the algorithm, the role of the programmer, and learning were examples of the construction of a structure on which it was possible to articulate the comparison of the three approaches we chose. Considering the metaphor of the process of weaving, these vocabularies are the warp, whose meaning is declined in the imperative procedural, declarative logic, and ML neural network approaches which in turn constitute the weft of the structure. From the combination of warp and weft, the epistemic insights emerge as overarching themes that allow disciplinary and experience-grounded knowledge to reach an epistemological depth.

Appendix 1

Annex 1. Code to program a machine playing Tic-Tac-Toe in Python

Note that, both due to time limitations of the teaching activities and of length limitations of the PythonTutor visualization tool, this program implements an "almost-perfect" strategy, as detailed in the comments of the code.

```
#check if player p
#wins on the board b
def wins(b, p):
  return ((b[6] == p and b[7] == p and b[8] == p)
  or (b[3] == p and b[4] == p and b[5] == p)
  or (b[0] == p and b[1] == p and b[2] == p)
  or  (b[6] == p and b[3] == p and b[0] == p)
  or  (b[7] == p and b[4] == p and b[1] == p)
  or (b[8] == p and b[5] == p and b[2] == p)
  or (b[6] == p and b[4] == p and b[2] == p)
  or (b[8] == p and b[4] == p and b[0] == p))

#returns next best move for player 'o'
#given the current state of the Board

def nextmove(Board):
  #Checks if 'o' can win with the next move
  for i in range(9):
    b = Board.copy()
    #if space i is empty
    if b[i] == '':
      #try to make the move
      b[i] = 'o'
      if wins(b,'o'):
        print("AI: win")
        return i

  #Otherwise, blocks 'x' to win with the next move
  for i in range(9):
    b = Board.copy()
    #if space i is empty
    if b[i] == '':
      #try to simulate 'x' move
      b[i] = 'x'
      if wins(b,'x'):
        print("AI: blocking opponent win")
        return i
```

```

  #Otherwise, try to make a move
  #that will allow me a fork
  for i in range(9):
    b = Board.copy()
    #if space i is empty
    if b[i] == '':
      #try to make the move
      b[i] = 'o'
      nvicmov = 0
      #check if with further moves
      #I will have more threats to win (fork)
      for j in range(9):
        sc2 = b.copy()
        if sc2[j] == '':
          sc2[j] = 'o'
          if wins(sc2,'o'):
            nvicmov += 1
          if nvicmov > 1:
            print("AI: fork")
            return i


  #Otherwise, try to avoid opponent (single) forks
  #Not perfect, should check also double forks
  for i in range(9):
    b = Board.copy()
    if b[i] == '': #if space i is empty
      b[i] = 'x' #try to make the moves
      nvicmov = 0
      #check if with further moves
      #opponent will have more threats to win (fork)
      for j in range(9):
        sc2 = b.copy()
        if sc2[j] == '':
          sc2[j] = 'x'
          if wins(sc2,'x'):
            nvicmov += 1
          if nvicmov > 1:
            print("AI: blocking a fork")
            return i
```

```
  #Otherwise I choose 'randomly',
  #but in the order center-corner-side

  #center
  if Board[4]=='':
    print("AI: center")
    return 4

  #corners
  for a in [0,2,6,8]:
    if Board[a]=='':
      print("AI: corner")
      return a

  #sides
  for l in [1,3,5,7]:
    if Board[l]=='':
      print("AI: side")
      return l


Board = ['x',  '',  '',
                '', 'o',  '',
                '', 'x',  '']

print(nextmove(Board))
```

Annex 2. Code to program a machine playing Tic-Tac-Toe in Prolog

We report the relevant part of the code we wrote in Prolog language, inspired by that formulated by Scott (2009, p. 564).

Knowledge base and definition of "equal" and "different":

```
% Let's teach to Prolog the basics (what it means that two variables are equal or different)
equal(A,A).
different(A,B) :- not(equal(A,B)).

% Winning combinations
win_line(6,7,8).
win_line(3,4,5).
win_line(0,1,2).
win_line(6,3,0).
win_line(7,4,1).
win_line(8,5,2).
win_line(6,4,2).
win_line(8,4,0).

% Of course, all the permutations are valid too: (1,2,3) or (3,1,2) doesn't change the result;)
tictactoe(A,B,C) :- win_line(A,B,C).
tictactoe(A,B,C) :- win_line(A,C,B).
tictactoe(A,B,C) :- win_line(B,A,C).
tictactoe(A,B,C) :- win_line(B,C,A).
tictactoe(A,B,C) :- win_line(C,A,B).
tictactoe(A,B,C) :- win_line(C,B,A).
```

Instruction to fill a cell and definition of "empty cell":

```
% A square is filled if it contains 'x' or 'o', empty otherwise
filled(A) :- x(A).
filled(A) :- o(A).
empty(A) :- not(filled(A)).
```

Rules and strategies:

```
% To decide the next move I apply the usual strategy
next_move(A) :- win(A).
next_move(A) :- prevent_other_win(A).
next_move(A) :- fork(A).
next_move(A) :- prevent_fork(A).
next_move(4).
next_move(0). next_move(2). next_move(6). next_move(8).
next_move(1). next_move(3). next_move(5). next_move(7).

%I win if I complete a line (I already have 'o' in B and C, so I put the third 'o' in A)
win(A) :- o(B), o(C), different(B,C), tictactoe(A,B,C).

%I prevent 'x' victory by choosing A
prevent_other_win(A) :- x(B), x(C), different(B,C), tictactoe(A,B,C).

%I create a fork if at the next move I can win in two moves
fork(A) :- o(B), o(C), different(B,C), tictactoe(A,B,D), tictactoe(A,C,E),  empty(D), empty(E).

%I prevent a fork if at the next move the opponent could win in two moves
prevent_fork(A) :- x(B), x(C), different(B,C), tictactoe(A,B,D), tictactoe(A,C,E), empty(D), empty(E).

%I make a move, of course if it's empty
computer_move :- next_move(A), empty(A), assert(o(A)).
```

Acknowledgements The authors thank all the partners of the I SEE project and, in particular, the teachers and researchers who have participated in the design and implementation of the module on Artificial Intelligence: Paola Fantini, Michela Clementi, Fabio Filippi, and Giovanni Ravaioli.

Funding Open access funding provided by Alma Mater Studiorum - Università di Bologna within the CRUI-CARE Agreement. Open access funding provided by Alma Mater Studiorum - Università di Bologna within the CRUI-CARE Agreement. The article reports a study carried out within the I SEE Project and re-implemented within the FEDORA project. I SEE has been funded with the support of the European Union and the Italian National Agency within the framework of the Erasmus+ Programme (Grant Agreement no. 2016-1-IT02-KA201-024373). FEDORA has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement no. 872841. M. Lodi's work has been supported by the Spoke 1 "FutureHPC & BigData" of the Italian Research Center on High-Performance Computing, Big Data and Quantum Computing (ICSC) funded by MUR Missione 4 Componente 2 Investimento 1.4: Potenziamento strutture di ricerca e creazione di "campioni nazionali di R&S (M4C2-19)" - Next Generation EU (NGEU).

Data Availability The data that support the findings of this study are openly available at https://iseeproject.eu/i-see-module-on-artificial-intelligence/.

Declarations

Competing Interests The authors declare no competing interests.

Disclaimer The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors; the Commission cannot be held responsible for any use which may be made of the information contained therein.

References

Alaa, A. M., & van der Schaar, M. (2019). Demystifying black-box models with symbolic metamodels. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32). Retrieved from https://proceedings.neurips.cc/paper_files/paper/2019/file/567b8f5f423af15818a068235807edc0-Paper.pdf

Anderson, C. (2008). *The end of theory: The data deluge makes the scientific method obsolete*. WIRED. Retrieved February 6, 2024, from http://www.wired.com/science/discoveries/magazine/16-07/pb_theory

Anegawa, S., Ho, I., Ly, K., Rounthwaite, J., & Migler, T. (2023). Learned monkeys: Emergent properties of deep reinforcement learning generated networks. In *Springer proceedings in complexity* (pp. 50–61). Springer International Publishing. https://doi.org/10.1007/978-3-031-28276-8_5

Barelli, E. (2022). *Complex systems simulations to develop agency and citizenship skills through science education*. Dissertation thesis, Alma Mater Studiorum Università di Bologna. Dottorato di ricerca in Data science and computation, 33 Ciclo. Retrieved from https://doi.org/10.48676/unibo/amsdottorato/10146

Berry, D. M. (2011). The computational turn: Thinking about the digital humanities. *Culture Machine, 12*, 1–22. Retrieved February 6, 2024, from https://culturemachine.net/wp-content/uploads/2019/01/10-Computational-Turn-440-893-1-PB.pdf

Billingsley, B. (2017). Teaching and learning about epistemic insight. *School Science Review, 365*, 59–64.

Billingsley, B., & Ramos Arias, A. (2017). Epistemic insight and classrooms with permeable walls. *School Science Review, 99*(367), 44–53.
Billingsley, B., Nassaji, M., Fraser, S., & Lawson, F. (2018). A framework for teaching epistemic insight in schools. *Research in Science Education, 48*(6), 1115–1131. https://doi.org/10.1007/s11165-018-9788-6
Boyd, D., & Crawford, K. (2012). Critical questions for big data. *Information, Communication and Society, 15*(5), 662–679. https://doi.org/10.1080/1369118x.2012.678878
Burrell, J. (2016). How the machine 'thinks': Understanding opacity in machine learning algorithms. *Big Data & Society, 3*(1), 205395171562251. https://doi.org/10.1177/2053951715622512
Carabantes, M. (2020). Black-box artificial intelligence: An epistemological and critical analysis. *AI & Society, 35*(2), 309–317. https://doi.org/10.1007/s00146-019-00888-w
Cilliers, P. (1998). *Complexity and postmodernism: Understanding complex systems*. Routledge.
Desai, J., Watson, D. I., Wang, V., Taddeo, M., & Floridi, L. (2022). The epistemological foundations of data science: A critical review. *Synthese, 200*(6). https://doi.org/10.1007/s11229-022-03933-2
Duit, R., Gropengießer, H., Kattmann, U., Komorek, M., & Parchmann, I. (2012). The model of educational reconstruction – A framework for improving teaching and learning science. In *SensePublishers eBooks* (pp. 13–37). SensePublishers. https://doi.org/10.1007/978-94-6091-900-8_2
Durán, J. M., & Jongsma, K. R. (2021). Who is afraid of black box algorithms? On the epistemological and ethical basis of trust in medical AI. *Journal of Medical Ethics*, 106820. https://doi.org/10.1136/medethics-2020-106820
Gabbrielli, M., & Martini, S. (2010). Programming languages: Principles and paradigms. *Undergraduate Topics in Computer Science*. https://doi.org/10.1007/978-1-84882-914-5
Goebel, R., Chander, A., Holzinger, K., Lecue, F., Akata, Z., Stumpf, S., Kieseberg, P., & Holzinger, A. (2018). Explainable AI: The new 42? In *Lecture Notes in Computer Science* (pp. 295–303). Springer Science+Business Media. https://doi.org/10.1007/978-3-319-99740-7_21
Gould, S. J. (1981). *The mismeasure of man*. W.W. Norton & Company.
Gunning, D., & Aha, D. W. (2019). DARPA's explainable artificial intelligence (XAI) program. *Ai Magazine, 40*(2), 44–58. https://doi.org/10.1609/aimag.v40i2.2850
Halevy, A., Norvig, P., & Pereira, F. L. (2009). The unreasonable effectiveness of data. *IEEE Intelligent Systems, 24*(2), 8–12. https://doi.org/10.1109/mis.2009.36
Hammoudeh, A., Tedmori, S., & Obeid, N. (2021). A reflection on learning from data: Epistemology issues and limitations. *arXiv preprint arXiv*. https://doi.org/10.48550/arXiv.2107.13270
Harman, G., & Kulkarni, S. (2007). *Reliable reasoning: Induction and statistical learning theory*. The MIT Press.
Hey, T., Tansley, S., & Tolle, K. (2009). *The fourth paradigm: Data-intensive scientific discovery*. Microsoft Research.
Hüllermeier, E., & Waegeman, W. (2021). Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine Learning, 110*(3), 457–506. https://doi.org/10.1007/s10994-021-05946-3
Ilkou, E., & Koutraki, M. (2020). Symbolic vs sub-symbolic AI methods: Friends or enemies? *Proceedings of the CIKM 2020 Workshops, October 19-20, Galway, Ireland.* Retrieved from https://ceur-ws.org/Vol-2699/paper06.pdf
Kitchin, R. (2014). Big Data, new epistemologies and paradigm shifts. *Big Data & Society, 1*(1), 205395171452848. https://doi.org/10.1177/2053951714528481
Kläs, M., & Vollmer, A. M. (2018). Uncertainty in machine learning applications: A practice-driven classification of uncertainty. In *Lecture Notes in Computer Science* (pp. 431–438). Springer Science+Business Media. https://doi.org/10.1007/978-3-319-99229-7_36
Leiter, C., Zhang, R., Chen, Y., Belouadi, J., Larionov, D., Fresen, V., & Eger, S. (2023). *ChatGPT: A meta-analysis after 2.5 months*. arXiv (Cornell University). https://doi.org/10.48550/arXiv.2302.13795
Leonelli, S. (2012). Introduction: Making sense of data-driven research in the biological and biomedical sciences. *Studies in History and Philosophy of Biological and Biomedical Sciences, 43*(*1*), 1–3.
Lodi, Michael, & Martini, Simone. (2021). Computational Thinking, Between Papert and Wing. *Science & Education, 30*(4), 883–908. https://doi.org/10.1007/s11191-021-00202-5
MacKenzie, D. (2001). *Mechanizing proof: Computing risk and trust*. MIT Press.
Monett, D., & Lewis, C. W. P. (2017). Getting clarity by defining artificial intelligence—A survey. In *Studies in applied philosophy, epistemology and rational ethics* (pp. 212–214). Springer. https://doi.org/10.1007/978-3-319-96448-5_21
Newell, A., & Simon, H. A. (1972). *Human problem solving*. Prentice-Hall.
Nilsson, N. J. (2010). *The quest for artificial intelligence*. Cambridge University Press.
O'Neil, C. (2016). *Weapons of math destruction, how big data increases inequality and threatens democracy*. Broadway Books.

Porway, J. (2014). *You can't just hack your way to social change*. Harvard Business Review. Retrieved February 6, 2024, from http://blogs.hbr.org/cs/2013/03/you_cant_just_hack_your_way_to.html

Prensky, M. (2009). H. sapiens digital: From digital immigrants and digital natives to digital wisdom. *Innovate: Journal of Online Education, 5*(3). Retrieved February 6, 2024, from https://www.learntechlib.org/p/104264/

Ravaioli, G. (2020). *Epistemological activators and students' epistemologies in learning modern STEM topics*. Doctoral dissertation. Alma Mater Studiorum Università di Bologna. Dottorato di ricerca in Fisica, 32 Ciclo. Retrieved from https://doi.org/10.6092/unibo/amsdottorato/9482

Ribes, D., & Jackson, S. J. (2013). Data bite man: The work of sustaining long-term study. In L. Gitelman (Ed.), *'Raw Data' is an Oxymoron* (pp. 147–166). MIT Press.

Robinson, J. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM, 12*(1), 23–41. https://doi.org/10.1145/321250.321253

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review, 65*(6), 386–408. https://doi.org/10.1037/h0042519

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

Rudin, C., & Wagstaff, K. L. (2014). Machine learning for science and society. *Machine Learning, 95*(1), 1–9. https://doi.org/10.1007/s10994-013-5425-9

Russell, S. J., & Norvig, P. (2021). *Artificial intelligence: A modern approach* (Fourth ed.). Pearson Education Limited.

Sætra, H. S. (2018). Science as a vocation in the era of big data: The philosophy of science behind big data and humanity's continued part in science. *Integrative Psychological and Behavioral Science, 52*(4), 508–522. https://doi.org/10.1007/s12124-018-9447-5

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development, 3*(3), 210–229.

Sanusi, I. T., Oyelere, S. S., Vartiainen, H., Suhonen, J., & Tukiainen, M. (2022). A systematic review of teaching and learning machine learning in K-12 education. *Education and Information Technologies, 28*(5), 5967–5997. https://doi.org/10.1007/s10639-022-11416-7

Scott, M. L. (2009). *Programming language pragmatics* (3rd ed.). Elsevier/Morgan Kaufmann Pub.

Shapiro, R. B., Fiebrink, R., & Norvig, P. (2018). How machine learning impacts the undergraduate computing curriculum. *Communications of the ACM, 61*(11), 27–29. https://doi.org/10.1145/3277567

Tedre, M., Toivonen, T., Kahila, J., Vartiainen, H., Valtonen, T., Jormanainen, I., & Pears, A. (2021). Teaching machine learning in K–12 classroom: Pedagogical and technological trajectories for artificial intelligence education. *IEEE Access, 9*, 110558–110572. https://doi.org/10.1109/access.2021.3097962

Touretzky, D. S., Gardner-McCune, C., Martin, F., & Seehorn, D. (2019). Envisioning AI for K-12: What should every child know about AI? *Proceedings of the AAAI Conference on Artificial Intelligence, 33*(01), 9795–9799. https://doi.org/10.1609/aaai.v33i01.33019795

Tsamados, A., Aggarwal, N., Cowls, J., Morley, J., Roberts, H., Taddeo, M., & Floridi, L. (2021). The ethics of algorithms: Key problems and solutions. *AI & Society, 37*(1), 215–230. https://doi.org/10.1007/s00146-021-01154-8

Valiant, L. (2013). *Probably approximately correct: Nature's algorithms for learning and prospering in a complex world*. Basic Books.

Van Zuylen, H. (2012). Difference between artificial intelligence and traditional methods. *Artificial Intelligence Applications to Critical Transportation Issues, E-C168*, 3–5.

Ventayen, R. J. M. (2023). OpenAI ChatGPT generated results: Similarity index of artificial intelligence-based contents. *Social Science Research Network*. https://doi.org/10.2139/ssrn.4332664

Wang, P. (2019). On defining artificial intelligence. *Journal of Artificial General Intelligence, 10*(2), 1–37. https://doi.org/10.2478/jagi-2019-0002

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

Wing, J. M. (2011). Research notebook: Computational thinking—what and why. In *The Link Magazine*. Carnegie Mellon University -- School of Computer Science. Retrieved February 6, 2024, from https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why

Zednik, C. (2021). Solving the black box problem: A normative framework for explainable artificial intelligence. *Philosophy & Technology, 34*(2), 265–288. https://doi.org/10.1007/s13347-019-00382-7

Zednik, C., & Boelsen, H. (2021). Preface: Overcoming opacity in machine learning. In C. Zednik & H. Boelsen (Eds.), *AISB 2021 Symposium Proceedings: Overcoming Opacity in Machine Learning*. Retrieved February 6, 2024, from http://www.explanations.ai/symposium/AISB21_Opacity_Proceedings.pdf

Zeng, D. (2013). From computational thinking to AI thinking [A letter from the editor]. *IEEE Intelligent Systems, 28*(6), 2–4. https://doi.org/10.1109/mis.2013.141