UNIVERSITÀ DEGLI STUDI DI UDINE
HIC SUNT FUTURA

DEPARTMENT OF MATHEMATICS, COMPUTER SCIENCE AND PHYSICS

PH.D. THESIS IN
COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE

Bridging Logic Programming and Deep Learning for Explainability through ILP

CANDIDATE
Talissa Dreossi

SUPERVISOR
Prof. Agostino Dovier

Cycle XXXVIII — A.Y. 2025-2026

Institute Contacts
Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Università degli Studi di Udine
Via delle Scienze, 206
33100 Udine — Italia
+39 0432 558400
https://www.dmif.uniud.it/

Author's Contacts
talissa.dreossi@uniud.it

Alle mie sorelle

Abstract

This thesis proposes a hybrid approach that bridges *Deep Learning* (DL) and *Logic Programming* (LP) to produce accurate but also interpretable *Artificial Intelligence* (AI) systems. While neural networks achieve outstanding predictive accuracy, their opaque nature limits transparency and accountability. This lack of transparency hinders their adoption in domains that demand accountability and interpretability, such as medical applications. To address this issue, the general idea we propose is to use the predictions generated by DL models as input to an *Inductive Logic Programming* (ILP) task. The ILP component then learns logical rules that explain the model's decisions in a form understandable to humans. In this framework, Deep Learning models, such as *You Only Look Once* (YOLO) for object detection, and *Convolutional Neural Network* (CNN) or *Recurrent Neural Network* (RNN) variants for classification and sequence tasks, are responsible for high-performance prediction, while ILP systems such as *Inductive Learning of Answer Set Programs* (ILASP) and *Fast Learning of Answer Set Programs* (FastLAS) are employed to extract symbolic rules. These systems induce sets of non-monotonic logical rules, expressed in *Answer Set Programming* (ASP), that capture relationships between input features and model predictions. The symbolic component enables the construction of interpretable explanations both in graphical form, via directed acyclic graphs generated by *eXplainable Answer Set Programming* (xASP), and textual form, facilitating translation into natural language.

The framework is applied and evaluated in three different domains: i) short-term meteorological forecasting, focusing on rainfall and lightning prediction; ii) juridical reasoning, based on the logical representation of *Italian Penal Code* articles and *Court of Cassation* decisions; and iii) veterinary medicine, automating biological image analysis for bull spermatozoa classification and morphological assessment. In each case, the hybrid models demonstrate competitive predictive accuracy while producing concise and semantically meaningful explanations. We also compare our explanations with common post-hoc methods, such as *Local Interpretable Model-agnostic Explanations* (LIME) and *SHapley Additive exPlanations* (SHAP), and *Large Language Model* (LLM) outputs.

Beyond the empirical evaluation, the thesis discusses several methodological challenges, including the scalability of ILP systems when applied to high-dimensional data. Overall, the research contributes a generalisable methodology for combining neural prediction with symbolic rule learning, offering a way towards interpretable, verifiable, and trustworthy AI. By demonstrating how ILP can formalise and explain complex neural reasoning processes, this work provides a step towards bridging data-driven learning with logic-based explanation.

Contents

List of Tables

List of Figures

I

Introduction

1 Introduction

Artificial intelligence has rapidly transformed various aspects of modern life, such as healthcare, finance, and education. Essentially, AI aspires to replicate, or at least simulate, human intelligence, extending the limits of what machines can achieve in terms of learning and reasoning. However, as AI systems become more powerful, a critical challenge emerges: balancing their predictive accuracy with interpretability. In other words, how can we build systems that not only perform well but also provide explanations for their behaviour? This question supports the motivation for the research presented in this thesis. Indeed, this thesis aims to examine the possibility to integrate DL and LP to lay the foundations for a new generation of AI systems. In particular, by combining *Artificial Neural Network* (ANN) with ILP, the objective is to build systems capable of making accurate predictions while also generating understandable rules that justify these predictions. DL models are used to process and analyse complex data, while ILP techniques are employed to derive logical rules that support the network's conclusions. This hybrid approach aims to bridge the gap between the high performance of deep learning models and the transparency of symbolic reasoning, advancing AI by providing interpretable and trustworthy applications.

The test cases examined in this thesis cover weather prediction, legal reasoning, and image recognition. Each of these domains can greatly benefit from explainable AI, as they often involve complex decision-making processes that can be made more transparent and interpretable, addressing challenges like accountability, trust, and ethical considerations.

1.1 Motivation

The field of AI has seen extraordinary growth over the last decade, with systems becoming more sophisticated and capable of performing tasks that were once considered impossible for machines. These advances are largely attributable to improvements in computing power, data availability, and new approaches to learning. With the advancement of AI systems, they are becoming more complex, not only in terms of their archi-

tectures but also in how they interact with the world around them. They have moved far beyond simple rule-based algorithms and now incorporate complex, multi-layered models able to learn from vast amounts of data. These systems often integrate diverse fields such as computer vision, natural language processing, or others. This fusion of disciplines allows AI to tackle more sophisticated tasks, such as autonomous driving, real-time language translation, and advanced healthcare diagnostics, by understanding and processing different types of data simultaneously. But, even if they become very powerful we are still trying to answer the question: "*Are they really smart?*". One can argue that, as Dijkstra said:

> *The question of whether a computer can think is no more interesting than the question of whether a submarine can swim.*
>
> – *Edsger Dijkstra*

Indeed, this question was one of those that originally gave birth to the field of artificial intelligence. Although the origins of AI is rather unclear, as it happens with many research fields, we can certainly state that Alan Turing was one of its founders. Even if he did not provide any algorithm in that sense, he raised the question of whether a system should be considered intelligent. He did this by formulating the so called *Imitation game* [1] in which a human interrogator must distinguish between a human and a machine through written dialogue. The test shifts the question of intelligence from defining it abstractly to evaluating whether a machine can convincingly imitate human behaviour.

As AI systems have grown more powerful, the Turing test can be considered passed. Anyway, their decision-making processes have also become more opaque, leading to concerns about transparency and trustworthiness. Indeed, the increased complexity of neural networks and other advanced models often makes it difficult to understand how these systems arrive at specific conclusions. We are no longer concerned with whether a computer can think, but rather with understanding the processes that lead to its specific outcomes. This is critical especially in domains like healthcare and finance, where the consequences of AI errors can be significant [2]. In response, there has been a growing focus on developing *Explainable Artificial Intelligence* (XAI) methods, which aim to make AI decisions more interpretable for human users.

The growing capabilities of AI systems have also raised important ethical questions, particularly concerning bias, fairness, and privacy. As AI models become more embedded in decision-making processes that affect millions of people, ensuring that these systems operate fairly and without discrimination has become a central challenge. However, if AI systems can be made explainable, then biases in their decision-making processes can be more easily identified and corrected.

1.1.1 Challenges

In this subsection, we will examine the challenges associated with the black-box nature of deep learning, which XAI seeks to overcome by enhancing model transparency and interpretability. The scenarios we are going to examine are not merely hypothetical. Indeed, these ethical dilemmas extend beyond transportation, military applications, and healthcare to various sectors. The decisions made by AI systems can significantly

impact individual lives and societal norms, necessitating a complete understanding of how these systems should be programmed to reflect ethical principles.

Bias Issues Bias issues occur when an AI system produces results that are systematically prejudiced due to erroneous assumptions in the machine learning process. This often originates from the data used to train AI models. In fact, they are often trained on historical data, which may already contain biases based on societal inequalities or discriminatory practices. This reinforcement of historical biases can occur across various sectors, from criminal justice to finance, where past decisions were influenced by prejudiced or unequal systems. For example, if a recruitment algorithm is trained on resumes from a predominantly male workforce, as it has happened in 2018 with Amazon's AI-powered recruitment tool [3], it may learn to associate male candidates with success, reinforcing gender biases in its future hiring recommendations. A similar issue arises in facial recognition systems. If these systems are primarily trained on images of lighter-skinned individuals, often due to an over-representation of such images, their performance may be biased. As a result, they may struggle to accurately identify individuals with darker skin tones, leading to biased and less reliable outcomes. In general, if the data used to train an AI system includes fewer examples of people from certain racial, gender, or socioeconomic backgrounds, the model may fail to recognize or accurately assess those groups. This imbalance skews the model's predictions in favour of the more represented groups, often leading to discriminatory effects in real-world applications. For instance, an AI system in healthcare may provide less accurate diagnoses for minority populations if it has been trained predominantly on data from a majority group, perpetuating inequalities in healthcare access and outcomes. This is what happened in the U.S. where healthcare systems have been reported to exhibit bias against people of colour [4].

Ethical Issues Artificial intelligence and autonomous systems present countless ethical challenges that require careful considerations. As example, let us consider one of the most prominent areas of concern, which is the decision-making processes of autonomous vehicles, particularly in scenarios where their actions may lead to harm. These ethical dilemmas raise fundamental questions about morality, responsibility, and the values that should guide AI behaviour. The *Autonomous Car Dilemma* exemplifies this challenge, where a self-driving car must make critical decisions in life-and-death situations:

An autonomous car must decide between running over a group of young children or swerving and hitting an elderly person. There are no other options. How should the car prioritize life, and who should program these decisions?

When faced with the choice of hitting a group of pedestrians, comprising children or elderly individuals, the car must investigate complex ethical considerations. Should it prioritize the lives of the younger children over the elderly, or should it attempt to minimize overall harm regardless of age? Similarly, imagine a military drone capable of autonomous decision-making using AI. This would allow the drone to make critical decisions about targeting and engagement under time constraints and incomplete information. For example, it may face a dilemma between striking a known terrorist target, risking civilian casualties, or withholding action, allowing the target to escape. On the other hand, in non-military contexts, AI systems designed for emotional support face

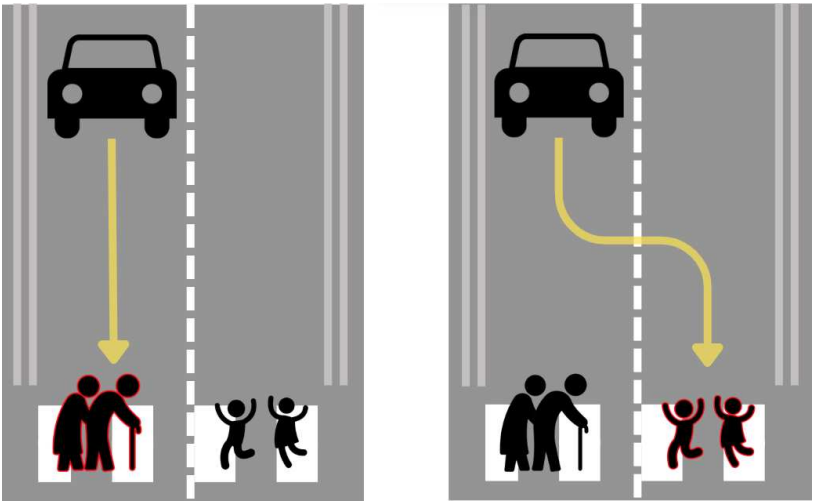


Figure 1.1: Representation of the autonomous dilemma.

ethical dilemmas such as deciding whether to breach confidentiality to alert authorities about a user at risk of self-harm, balancing privacy against potential lives saved.

Explainability alone cannot dictate the "right" ethical solution; it must be paired with robust ethical frameworks, stakeholder engagement, and societal consensus. Even if XAI is not a complete solution to such ethical dilemmas, it offers important advantages. For example, in the context of the car dilemma, understanding why an AI prioritised one action over another can help stakeholders assess whether the system aligns with societal and ethical norms. Moreover, it becomes easier to identify biases, errors, or undesirable behaviour if we know the reason behind a decision.

Other ethical problems arise from the fact that the lack of transparency can create opportunities for misuse and exploitation. Without a clear understanding of how decisions are made, malicious actors can exploit these systems to achieve harmful objectives. For example, in marketing or social media algorithms, non-explainable AI may inadvertently promote biased or harmful content, as users and regulators cannot trace the decision-making process back to its source. In security contexts, autonomous systems like drones that make decisions without explanation may be manipulated to carry out unethical or unlawful actions.

Accountability Accountability in AI systems is also critical. When decisions are made by a non-explainable AI, determining who is responsible for the outcomes becomes extremely difficult. If an AI system makes a decision that leads to adverse consequences, such as denying a loan or misdiagnosing a patient, there is often no clear pathway to identify who should be held accountable: the developers, the data scientists, or the organisation deploying the AI? This lack of accountability raises important ethical and legal concerns. For example, in a scenario where a non-explainable AI system is used in judicial decisions, the inability to understand how the AI reached its conclusions can undermine the fairness and transparency of the legal process. Furthermore, regulatory

bodies may struggle to enforce accountability measures if they cannot establish how decisions are made. This situation creates a precarious environment where harmful decisions can be made, leading to less public trust in AI technologies.

Additionally, there are problems also in the field of copyrights. Non-explainable systems that produce artworks, music, or text make it difficult to assign intellectual property rights. Although creators, developers, and organisations may seek ownership of AI-generated content, the lack of transparency in the model's decision-making process makes it difficult to assess originality or assign responsibility. Even if this is not a possibly harmful problem, it still is an issue that must be addressed.

Debugging and Improvement Effective debugging and improvement of AI models are heavily reliant on understanding their inner workings. Since AI systems often function as black boxes, where the inputs and outputs are visible, but the processes in between remain obscured, it is challenging for developers to diagnose issues, identify biases, or rectify errors. For instance, if an AI system consistently misclassifies certain data points, developers may struggle to understand the underlying cause: are there biases in the training data or is there a flaw in the model architecture? Without the ability to trace the decision-making process, addressing these issues becomes a highly uncertain and inefficient job. As a result, the model may remain flawed over time, perpetuating errors and biases, which eventually impacts its effectiveness and reliability.

Regulatory Compliance Many industries are governed by strict regulations that mandate transparency in decision-making processes. For instance, in finance, the *Fair Credit Reporting Act* requires lenders to provide reasons for denying credit, and in healthcare, the *Health Insurance Portability and Accountability Act* (HIPAA) emphasises the importance of maintaining patient confidentiality and providing clear communication regarding medical decisions. If an organisation employs an AI system that cannot articulate its decision-making process, it may struggle to provide the required explanations to regulatory bodies, risking legal repercussions, fines, and damage to reputation. This regulatory landscape forces organisations to either renounce potentially beneficial AI technologies or invest in supplementary measures to ensure compliance, thereby complicating the integration of AI into existing frameworks.

1.1.2 EU AI Act

As part of its full AI strategy, the European Union has introduced the *EU AI Act*[1], a groundbreaking regulation aimed at ensuring the development and use of artificial intelligence in a safe, reliable, and transparent manner. This legislation is the world's first legal framework designed to regulate AI across member states and categorizes AI systems based on their associated risks, which has deep implications for organizations involved in AI development or deployment within the EU. The Act seeks to guarantee that AI systems operate safely, supporting existing fundamental rights, and aligning with the EU's core values.

[1]Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence and amending Regulations (EC) No 300/2008, (EU) No 167/2013, (EU) No 168/2013, (EU) 2018/858, (EU) 2018/1139 and (EU) 2019/2144 and Directives 2014/90/EU, (EU) 2016/797 and (EU) 2020/1828 (Artificial Intelligence Act)

The Act defines in particular some requirements for AI to be considered trustworthy. They are formulated as *conformity assessment* (CA), defined in Article 43, and are fundamental for the process of verifying and/or demonstrating that a high-risk system complies with certain requirements laid out in the Act. These assessments are:

1. *Risk management system*: it must be developed a continuous, iterative process to identify and mitigate potential threats to health, safety, and human rights throughout a system's entire lifespan. Providers must conduct rigorous testing against specific metrics to ensure their technology functions reliably and safely before reaching the public.

2. *Data governance*: AI systems must be developed using relevant, representative, and error-free datasets that are subject to rigorous governance and management practices, including bias detection, data preparation, and strict security safeguards for sensitive personal information, to ensure they meet specific quality criteria appropriate for their intended purpose.

3. *Technical documentation*: a technical documentation must be established and kept up-to-date and is intended to demonstrate the system's compliance with regulatory requirements while providing authorities and notified bodies with clear information for assessment.

4. *Record keeping*: AI systems shall technically allow for the automatic recording of events, *i.e.* logs, over the lifetime of the system.

5. *Transparency and provision of information*: AI systems shall be designed and developed in such a way as to ensure that their operation is sufficiently transparent to enable deployers to interpret a system's output and use it appropriately.

6. *Human oversigh*t: AI systems shall be designed and developed in such a way that they can be effectively overseen by natural persons during the period in which they are in use.

7. *Accuracy, robustness, and cybersecurity*: AI systems must be designed to achieve an appropriate level of accuracy, robustness, and cybersecurity throughout their lifecycle. The levels of accuracy and the relevant accuracy metrics of high-risk AI systems shall be declared in the accompanying instructions of use.

Effective from August 2024, the Act imposes specific compliance timelines: within six months, organizations must adhere to regulations concerning prohibited practices, while obligations for general-purpose AI models will take effect after twelve months. High-risk AI systems will face additional obligations within twenty-four months. The EU AI Act employs a risk-based approach, classifying AI systems into four categories: *unacceptable*, *high risk*, *limited risk*, and *minimal risk*. Unacceptable AI applications, which contradict EU values and rights, are prohibited outright. High-risk systems, particularly in sectors like finance and healthcare, must meet stringent requirements, including comprehensive quality and risk management processes, conformity assessments through self-evaluation and third-party audits, and mandatory registration in a public EU database. Limited-risk applications have transparency obligations, while minimal-risk systems encourage ethical use without stringent regulatory constraints.

1.2 Related Works and History

Inductive Logic Programming was pioneered by Muggleton [5], who formally defined ILP as a discipline concerned with the inductive construction of first-order clausal theories from examples and background knowledge. Traditional ILP approaches typically follow either a top-down or bottom-up search strategy, relying on the assumption that atoms are strictly true or false. This binary perspective limits the system's ability to handle imprecise or uncertain data, which is common in real-world applications. More recent ILP research has introduced meta-level approaches, such as the method proposed by Inoue et al. [6], where the ILP problem is reformulated as a meta-level logic program. This allows the system to reason about the structure and properties of programs, offering notable advantages in terms of learning efficiency, particularly for recursive and optimal programs [7, 8, 9, 10]. Given the inherent noise in real-world data, modern ILP methods incorporate techniques to mitigate uncertainty. One common strategy involves optimizing program selection to maximise coverage of positive examples while minimising negative ones. Another approach, introduced by Cropper et al. [11], focuses on minimising the difference between learned hypotheses and a target solution. Additionally, *Statistical Relational AI* (StarAI) [12, 13] extends ILP by integrating probabilistic reasoning, using probabilities or weights to express confidence levels in learned rules. These advancements enhance ILP's applicability to complex domains where strict logical assumptions are impractical.

The ILP systems employed in this thesis are ILASP [14] and FastLAS [15], which integrate Answer Set Programming as learning framework dealing also with noisy data by assigning weights for examples. These approaches learn ASP programs rather than traditional logic programs, exploiting the expressiveness of ASP. Unlike definite logic programs (e.g., Prolog), which have a single least Herbrand model (see Chapter 2), ASP programs may have one, multiple, or no *stable models* (*answer sets*). This non-monotonic nature makes ASP particularly suited for modelling common-sense reasoning and handling dynamic scenarios. Additionally, ASP extends standard logic programming with features such as disjunctions in clause heads, choice rules, and hard or weak constraints, enhancing its flexibility in reasoning tasks. Despite significant progress, ILP remains underused in real world applications, even though tools such as ILASP and FastLAS have demonstrated their ability to produce interpretable, rule-based models [8, 16, 17, 18, 19, 20]. Notably, Cunnington et al. [21] proposed a hybrid approach that integrates neural networks with symbolic reasoning, exploiting the strengths of both paradigms. By combining the predictive power of neural networks with the logical transparency of symbolic methods, this approach improves both explainability and robustness without sacrificing performance.

Explainable AI techniques purpose is to generate human-readable explanations that enable users to understand and trust the outcomes produced by deep learning models. Regulatory frameworks such as the *European General Data Protection Regulation* (GDPR) [22] have reinforced the need for XAI research, emphasising critical aspects such as ethical considerations, justification, trust, and bias mitigation [23, 24, 25, 26] to ensure the development of reliable and transparent AI systems. In the context of this thesis, our focus is on using ASP to explain the behaviour of deep learning systems. Explainability through ASP was indeed subject of ongoing research by Alviano et al. [27] who demonstrated how the inclusion or exclusion of atoms in answer sets can

be linked to the underlying logic rules through visualization techniques such as *Directed Acyclic Graph* (DAG). Another notable tool is the causal graph framework introduced by Cabalar et al. [28], where rules are represented as nodes and their application order as edges, offering a clear depiction of logical dependencies.

1.3 The Role of ILP

In this section, we motivate the use of ILP by highlighting its key advantages over conventional machine learning approaches.

The first aspect to consider is that ILP's hypotheses are expressed as set of logic rules, which can be naturally translated into *Natural Language* (NL). Indeed, unlike most machine learning approaches, which typically output numerical predictions or opaque functions, ILP induces a set logical rules, *i.e.* a logic program. This means its predictions are logical assertions rather than numerical values, which can offer greater interpretability.

Another distinctive advantage of ILP lies in its treatment of the cost function during learning. Traditional machine learning models typically minimise a loss function, such as mean squared error for regression or cross-entropy for classification, that measures the discrepancy between predicted and true labels. While effective in many contexts, this optimisation often focuses solely on reducing training error, which can lead to overfitting, especially in the presence of noise [29]. In contrast, modern ILP systems adopt a different learning paradigm. Indeed, systems like FastLAS, as we will see in detail in Section 3.4.3, allows to define a cost function that is based on the specific predicates used in the learned rules. This allows the learning process to be guided not solely by predictive accuracy, but by the structure and content of the hypotheses themselves effectively prioritising certain features or knowledge elements over others. Moreover, they search for hypotheses that are globally optimal with respect to a symbolic cost function, aiming to cover all positive examples, exclude all negative ones, and minimise the complexity of the resulting rule set [30]. This approach promotes better generalisation and allows for the learning of concise, logically consistent models within a well-defined hypothesis space.

The generalisation is also permitted by the possibility to deal with noisy data. Indeed, if the algorithm tries too hard to perfectly fit every single training example, it might end up learning the random errors or irrelevant patterns (noise) present in those examples, rather than the underlying general relationships. This "memorization" of noise means that the hypothesis will not perform well when encountering new, unseen data that does not contain the same noise. In this case, ILASP and FastLAS allow the assignment of penalties to individual examples. A higher penalty reduces the priority of covering that example during learning, meaning the system will prefer hypotheses that cover lower-penalty examples while potentially excluding those associated with higher costs.

As will be shown in Part III, ILP can achieve results comparable to, or even exceeding, those of general machine learning techniques, while requiring smaller amounts of training data.

In relation to the CAs, defined in Section 1.1.2, ILP is able to answer many of them. In particular:

- Data governance: ILP naturally encourages structured, semantically meaningful

data representations, often grounded in background knowledge and domain ontologies.

- Technical documentation: ILP models are self-documenting by design since hypotheses are explicit logical rules, assumptions are captured as background knowledge, and learning settings and constraints are declarative and reproducible.
- Transparency and provision of information: this is a core strength of ILP, indeed outputs are symbolic, human-readable rules, reasoning chains are explicit, and predictions can often be traced back to specific clauses and facts.
- Human oversight: humans can inspect, modify, and constrain hypotheses, moreover background knowledge can encode expert rules.

Regarding the remaining CAs, ILP shows some limitations:

- Risk management system: ILP does not itself define a continuous risk management process, nor does it mandate lifecycle-wide testing or monitoring. These aspects must be implemented at the system and organisational level.
- Record keeping: automatic event logging over the entire system lifetime is not intrinsic to ILP. Logging capabilities depend on the surrounding infrastructure rather than the learning paradigm itself.
- Accuracy, robustness and cybersecurity: the accuracy of ILP is not so high and robustness depends on the entire system while cybersecurity is not addressed by ILP itself.

1.4 Structure of the Thesis

The thesis is organised into four main parts, each designed to progressively introduce the theoretical foundations, methodological approaches, and practical contributions of this research. In particular:

- *Part I – Introduction*: this part outlines the motivation and research questions that underpin the thesis. It discusses the limitations of opaque, black-box models in critical domains and motivates the use of ILP as a means to enhance interpretability in machine learning systems. Furthermore, it situates the research within the existing literature and provides a high-level justification for the adopted hybrid approach combining neural models with logic-based explanations.
- *Part II – Background*: this part presents the theoretical background required to understand the methods and techniques employed in the thesis. It begins with a detailed exposition of logic programming, including first-order logic, Answer Set Programming, and non-monotonic reasoning. The fundamentals of ILP are then introduced, with particular emphasis on the ILASP and FastLAS frameworks. Additionally, this part reviews relevant concepts from deep learning, including neural architectures such as CNNs and YOLO, and surveys the state of the art in explainable AI. The discussion includes the motivation for XAI, existing taxonomies

of explanation methods, and the specific role of symbolic knowledge extraction in contrast to post-hoc, model-agnostic techniques. The benefits and limitations of different XAI paradigms are critically analysed, and the relevance of symbolic approaches for generating faithful and comprehensible explanations is highlighted.

- *Part III – Contributions*: this part presents our contributions with the practical implementation and evaluation of the proposed methodology across three diverse application domains, which are weather forecasting, juridical reasoning, and veterinary medicine. Each chapter describes the domain-specific problem, the available data, and the design of the hybrid learning system. The results are assessed both in terms of predictive accuracy and the quality of the symbolic explanations. Particular emphasis is placed on how the explanations contribute to transparency, trustworthiness, and potential user understanding within each application context. Furthermore, this part also includes a chapter (Chapter 10) which explores the use of logic programming as a tool to support students' preparation for exams.

- *Conclusions*: the final part synthesises the main findings of the thesis and reflects on its theoretical and practical contributions. It discusses the benefits and limitations of the proposed approach, especially with regard to explainability, scalability, and domain generality.

II
Background

2

Logical Background

To effectively model specific tasks, artificial intelligence relies on structured representations of knowledge that a computer can process and, somehow, understand. This representation should enable automated reasoning, facilitating the agent's ability to interpret, infer, and act based on the encoded information. Indeed, *Knowledge Representation* (KR) is the discipline dedicated to encoding the beliefs, intentions, and judgements of an intelligent agent in a symbolic, modular, and transparent manner.

2.1 Introduction

Fundamentally, knowledge representation relies on atomic symbols, each tied to a specific meaning, which can be organised into structures reflecting particular beliefs or decisions. Among the various approaches to knowledge representation, logic programming plays a central role, particularly within *rule-based* systems. In this paradigm, knowledge is expressed in the form of logical rules, often resembling `IF ... THEN ...` structures. Logic programming systems employ symbolic representations, ensuring that rules and deductions remain both interpretable and easily translatable into natural language. This structured approach enhances transparency and helps human understanding of the reasoning process. Notable logic programming paradigms, all based on the notion of *clause*, that use a rule-based approach are *Prolog* [31], ASP [32, 33], and *Datalog* [34].

Since LP inherit the basis from *First Order Logic* (FOL), this chapter begins with a brief overview of FOL, followed by an in-depth exploration of LP and its principles.

2.2 First-Order Logic

Unlike propositional logic, which deals with simple `true` or `false` statements, first-order logic allows for quantification over objects through the use of variables, predicates, and quantifiers like *for all* ($\forall$) and *exists* ($\exists$). Moreover, it also deals with *predicates*. For example: "8 is an even number" could be translated into $e(c)$, where e is a *predicate symbol* denoting the even property, and c is a *constant symbol*, denoting the number

8 which is even. This expressive power makes FOL suitable for representing complex structures and reasoning about them.

2.2.1 Syntax

A FOL language can be defined univocally by its *signature* [35].

Definition 2.2.1 (Signature)**.** *A FOL language is univocally characterised by a* signature Σ *which consists of:*

- Π*: a set of predicate symbols* $(p_1, p_2, \dots)$*,*
- $\mathcal{F}$*: a set of function symbols* $(f_1, f_2, \dots)$ *and constant symbols* $(c_1, c_2, \dots)$*,*
- $\mathcal{V}$*: an infinite set of variable symbols* $(v_1, v_2, \dots)$*.*

In every language, logic connectives $(\wedge, \vee, \neg, \rightarrow, \leftrightarrow)$ and quantifiers $(\forall, \exists)$ are included. Each symbol of the signature is denoted by its *arity*, *i.e.* the number of its arguments, determined by the arity function.

Definition 2.2.2 (Arity Function)**.** *The arity function* $ar : \Sigma \rightarrow \mathbb{N}$ *of a symbol is defined such that:*

- $ar(X) = 0$ *for each* $X \in \mathcal{V}$*,*
- $ar(c) = 0$ *for each constant symbol* $c \in \mathcal{F}$*,*
- $ar(f) > 0$ *for each* $f \in \mathcal{F}$*,*
- $ar(p) \geq 0$ *for each* $p \in \Pi$*.*

We now introduce several definitions to clarify key terms that will be used in the next section and throughout this thesis.

Definition 2.2.3 (Term)**.** *A* term *is recursively defined as:*

- *variables and constants are terms,*
- *if* $t_1, t_2, \dots t_n$ *are terms and* $f \in \mathcal{F}$ *with* $ar(f) = n$*, then* $f(t_1, t_2, \dots t_n)$ *is a term.*

Given a term t, the set of variables that occur in t is denoted by $vars(t)$. When $vars(t) = 0$, t is said to be a *ground* term.

Definition 2.2.4 (Atom)**.** *Given* $p \in \Pi$*, with* $ar(p) = n$*, and* $t_1, \dots, t_n$ *terms,* $p(t_1, \dots, t_n)$ *is called* atom *(or* atomic formula*). If* $t_1, \dots, t_n$ *are ground then it is a ground atom (or a ground atomic formula).*

Definition 2.2.5 (Literal)**.** *A* literal *is either an atomic formula (positive literal) or the negation of the atomic formula itself (negative literal).*

Definition 2.2.6 (Formula)**.** *A formula is inductively defined as follows:*

- *an atomic formula is a formula;*

- *if ϕ is a formula, then $\neg\phi$ is a formula;*
- *if ϕ and ψ are formulas, then $\phi \vee \psi$ is a formula;*
- *if ϕ is a formula and $X \in \mathcal{V}$, then $\exists X\phi$ is a formula.*

All the other logical connectives ($\wedge, \rightarrow, \leftrightarrow, \forall$) can be obtained from $\vee$ (e.g. $\phi \wedge \psi$ stands for $\neg(\neg\phi \vee \neg\psi)$).

Definition 2.2.7. *We say that a variable X occurs free in a formula φ if one of the following conditions holds:*

- *φ is an atomic formula of the form $p(t_1, \ldots, t_n)$ and $X \in vars(t_1) \cup \cdots \cup vars(t_n)$.*
- *φ is of the form $\neg\psi$ and X occurs free in ψ.*
- *φ is of the form $\psi \vee \eta$ and X occurs free in ψ or X occurs free in η.*
- *φ is of the form $\exists Y\, \psi$ if X occurs free in ψ and $X \neq Y$.*

If φ is a formula, then $vars(\varphi)$ denotes the set of variables that occur free in φ. If $vars(\varphi) = \emptyset$, then φ is called a *statement*.

2.2.2 Semantics

The semantics of FOL defines how formulas are interpreted and assigned truth values. An interpretation consists of a *domain*, which is the set of all possible objects under consideration, and an *interpretation function* that assigns possibly infinite meaning to symbols. Specifically, it maps each constant to an object in the domain, each predicate to a relation among objects, and each function symbol to a function over the domain.

Definition 2.2.8 (Interpretation)**.** *An* interpretation $\mathcal{A} = \langle A, (\cdot)^{\mathcal{A}} \rangle$ *for a signature Σ consist of:*

- *a non-empty set A called domain,*
- *a function $(\cdot)^{\mathcal{A}}$ such that there exists:*
  - *an element $(c)^{\mathcal{A}} \in A$, for each constant symbol $c \in \mathcal{F}$,*
  - *an n-ary function $(f)^{\mathcal{A}} : A^n \rightarrow A$, for each function symbol $f \in \mathcal{F}$,*
  - *an n-ary predicate $(p)^{\mathcal{A}} \subseteq A^n$, for each predicate symbol $p \in \Pi$.*

In the case of terms containing variables, the interpretation is also called *assignment*.

Definition 2.2.9 (Assignment)**.** *Given an interpretation $\mathcal{A} = \langle A, (\cdot)^{\mathcal{A}} \rangle$, a term t, and a set of variables B such that $B \supseteq vars(t)$, an* assignment *is a function $\sigma : B \rightarrow A$ that maps variables in B to elements of the domain A. The* valuation *of the term t under this assignment is then defined as follows:*

$$(t\sigma)^{\mathcal{A}} = \begin{cases} \sigma(X) & \text{if } t \equiv X \text{ is a variable in } \mathcal{V} \\ (c)^{\mathcal{A}} & \text{if } t \equiv c \text{ is a symbol in } \mathcal{F}, \text{ with } ar(c) = 0 \\ (f)^{\mathcal{A}}((t_1\sigma)^{\mathcal{A}} \ldots, (t_n\sigma)^{\mathcal{A}}) & \text{if } t \text{ is in the form } f(t_1, \cdots, t_n) \end{cases} \quad (2.1)$$

The following definition allows us to define the *truth value* of an atomic formula.

Definition 2.2.10 (Truth value)**.** *Let $\mathcal{A}$ be an interpretation and $p(t_1, \cdots, t_n)$ an atomic formula. Given a set of variables B, such that $B \supseteq vars(p(t_1, \cdots, t_n))$, and the assignment $\sigma : B \to A$, where A is the domain, then the* truth value *of the atomic formula is:*

$$V^{\mathcal{A}}(p(t_1, \cdots, t_n)) = \begin{cases} \texttt{true} & \text{if } \langle (t_1\sigma)^{\mathcal{A}}, \cdots, (t_n\sigma)^{\mathcal{A}} \rangle \in (p)^{\mathcal{A}} \\ \texttt{false} & \text{otherwise} \end{cases} \tag{2.2}$$

We denote that an atomic formula ψ is `true` *wrt* the interpretation $\mathcal{A}$ using the notation $\mathcal{A} \models \psi$.

Definition 2.2.11 (Satisfiability)**.** *Given a statement φ we say that:*

- *φ is* satisfiable *if there exists an interpretation $\mathcal{A}$ such that $\mathcal{A} \models \varphi$,*
- *φ is* unsatisfiable *if there does not exist any interpretation $\mathcal{A}$ such that $\mathcal{A} \models \varphi$,*
- *φ is* valid *if for all interpretations $\mathcal{A}$ it holds that $\mathcal{A} \models \varphi$.*

Definition 2.2.12 (Theory)**.** *Given a set of statements Γ (also called a theory), we say that:*

- *Γ is* satisfiable *if there exists an interpretation $\mathcal{A}$ such that $\mathcal{A} \models \varphi$ for every $\varphi \in \Gamma$; otherwise, Γ is said to be* unsatisfiable *(or* inconsistent*);*
- *a formula φ is a logical consequence of Γ if, for every model $\mathcal{A}$ of Γ, it holds that $\mathcal{A} \models \varphi$. By abuse of notation, this is denoted as $\Gamma \models \varphi$.*
- *If $\Gamma = \emptyset$, instead of writing $\Gamma \models \emptyset$, we simply write $\models \varphi$. The notation $\models \varphi$ therefore expresses the validity of the formula φ.*

From the above definitions, we can enunciate the following lemma.

Lemma 2.2.13. *Let Γ be a set of statements and φ a statement, then the following holds:*

$$\Gamma \models \varphi \text{ if and only if } \Gamma \cup \{\neg\varphi\} \text{ is unsatisfiable.}$$

Definition 2.2.14 (Model)**.** *A* model *of a formula φ is an interpretation $\mathcal{A}$ that satisfies φ. The notion can be extended to set of formulas.*

2.3 Logic Programming

In this section, we deepen into the fundamentals of logic programs, first focusing on *definite* and then on *general programs*. Definite programs operate within the domain of *monotonic* logic, meaning that the conclusions derived from a given set of premises remain valid even if new information is added. However, while monotonic logic is effective for certain applications, it has limitations when dealing with incomplete or evolving information. To address these limitations, we examine general programs, which expand the expressiveness of logic programming by allowing for both positive and negative information. This shift introduces *non-monotonic* reasoning, enabling programs to retract conclusions in light of new data.

2.3.1 Definite Programs

A *logic program* is defined as a conjunction of a set of rules. Let us define precisely rules (that are first-order formulas).

Definition 2.3.1 (Rule)**.** *A definite rule is an expression of the form:*

$$H \leftarrow A_1, \ldots, A_n.$$

where H is an atom, called the head *of the rule and $A_1, \ldots, A_n$ are atoms, called the* body *of the rule. When $n = 0$, i.e. the rule consists only of the head, the expression is said to be a* fact*. If instead we have a rule with no head,* i.e. *$\leftarrow A_1, \ldots, A_n$. then it is called* goal*. Furthermore, if $n = 0$ we denote the empty goal as $\leftarrow \square$.*

The $\leftarrow$ must be considered as the implication, but with opposite direction. This means that if the body is true, then also the head should be true. We will denote the literals in the head of a rule R as $head(R)$, and the set of literals in the body as $body(R)$.

In any rule, the comma represents the logical conjunction and, since it is known that $P \rightarrow Q \equiv \neg P \vee Q$, then a rule can be rewritten as $H \vee \neg A_1 \vee \cdots \vee \neg A_n$. A disjunction of literals is called a *clause*, therefore we can say that logic programs are set of clauses.

Definition 2.3.2 (Clause)**.** *A disjunction of literals is called* clause*. If the clause has no negative literals and:*

- *at most one positive literal, then it is called* Horn clause*,*
- *exactly one positive literal, then it is called* definite clause*.*

In Definition 2.2.12, a theory is defined as a set of formulas. When these formulas take the form of clauses, it is common to refer to theory with *program*. If a program is composed of only definite clauses then it is called *definite program*.

To understand a logic program, we must either identify a model for the program or determine that no such model exists. Since an interpretation of a logic program can vary, there are potentially infinite interpretations that arise simply from renaming elements within the domain. The concept of the *Herbrand universe* addresses this by limiting our focus to a single, specific domain constructed from the program's own syntax, thereby eliminating this infinite variability.

Definition 2.3.3 (Herbrand Universe)**.** *Given the alphabet $\Sigma = (\mathcal{F}, \Pi, \mathcal{V})$, the set of all ground terms $\mathcal{T}(\mathcal{F})$ that can be formed using $\mathcal{F}$ is called* Herbrand Universe *of Σ.*

Example 2.3.1
Consider the signature $\Sigma = (\mathcal{F}, \Pi, \mathcal{V})$, with $\mathcal{F} = \{1, p\}$ $(ar(p) = 1)$. The Herbrand Universe is the following: $T(\mathcal{F}) = \{1, p(1), p(p(1)) \ldots\}$.

We can now define what a Herbrand interpretation is.

Definition 2.3.4 (Herbrand Interpretation)**.** *Given the alphabet $\Sigma = (\mathcal{F}, \Pi, \mathcal{V})$, the* Herbrand Interpretation *$\mathcal{H} = \langle \mathcal{T}(\mathcal{F}), (\cdot)^{\mathcal{H}} \rangle$ has:*

- *as domain the Herbrand Universe* $\mathcal{T}(\mathcal{F})$,
- *as function* $(\cdot)^{\mathcal{H}}$ *the function defined as follows*

$$\begin{cases} c^{\mathcal{H}} = c \\ f^{\mathcal{H}}(t_1, \ldots, t_n) = f(t_1^{\mathcal{H}}, \ldots, t_n^{\mathcal{H}}) \end{cases} \tag{2.3}$$

The Herbrand Interpretation is, in other words, the function that assigns every ground term into itself.

The concept of the *Herbrand base* will also be useful for assigning semantics to programs.

Definition 2.3.5 (Herbrand Base)**.** *Given the alphabet* $\Sigma = (\mathcal{F}, \Pi, \mathcal{V})$*, the set of all ground atoms* $\mathcal{B}_{\Pi,\mathcal{F}} = \{p(t_1, \ldots, t_n) : p \in \Pi, t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})\}$ *is called* Herbrand Base.

Definition 2.3.6 (Herbrand interpretation)**.** *An* Herbrand interpretation I *is a Herbrand model for a theory* T *if* $I \models T$.

Note that we will denote a particular Herbrand interpretation simply by specifying the set of atoms that are true.

We can now formally state the following theorem. For this and the followings, proofs can be found in [36].

Theorem 2.3.7. *Given a definite program* P*, the Herbrand base* B_P *is a model for* P.

For B_P to be a model of P, every clause in P should be true under B_P. There exist two possible scenarios:

1. the clause in P is a fact, then it is already in B_P by definition of the Herbrand base;
2. the clause in P is a rule and so it is satisfied under B_P if, whenever the body is true in B_P, the head is also true. The construction of the least Herbrand model ensures that this is the case.

For logic programs in general, not only for definite ones, the following theorem and corollary hold.

Theorem 2.3.8. *Let* P *be a logic program. Then* P *has a model if and only if* P *has a Herbrand model.*

Corollary 2.3.9. *Let* P *be a logic program, and* $A \in B_P$ *a ground atom. Then* $P \models A$ *(*A *is a logical consequence of* P*) if and only if* A *is true in all Herbrand models of* P.

It can be demonstrated that, to compute the minimum Herbrand model M_P of a definite ground program P, it is possible to use a bottom-up approach with repeated applications of the operator T_P, defined below.

Definition 2.3.10 (T_P)**.** *Let* I *be a Herbrand interpretation for a definite program* P*. The immediate consequence of* I *is defined as:*

$$T_P(I) = \{H \mid H \leftarrow B_1, \dots B_n \in ground(P) \wedge \{B_1, \dots, B_n\} \subseteq I\}$$

The idea is to apply T_P starting from an empty interpretation and stopping at a fixpoint.

Definition 2.3.11. *Let P be a definite program. $T_P \uparrow^i$ is defined as follows:*

$$\begin{aligned} T_P \uparrow^0 &= \emptyset \\ T_P \uparrow^{n+1} &= T_P(T_P \uparrow^n) \\ T_P \uparrow^\omega &= \bigcup_{n \geq 0} T_P \uparrow^n \end{aligned}$$

Formally, the following theorems hold.

Theorem 2.3.12. *Let P be a definite clause program. Then T_P admits a fixpoint, i.e. a Herbrand interpretation F such that $T_P(F) = F$. Moreover, there exists a unique fixpoint that is minimal with respect to set inclusion, denoted $lfp(T_P)$.*

Theorem 2.3.13. *Let P be a definite clause program and M_P the minimum Herbrand model, then the following equivalence holds:*

$$M_P = lfp(T_P)$$

If the Herbrand Universe is finite, this algorithm computes M_P in polynomial time relative to the number of rules in P. Indeed, the worst-case scenario involves $|B_P|$ applications of T_P, each potentially requiring scanning the entire program P to add a single atom to M_P.

2.3.2 General Programs

In *monotonic reasoning*, conclusions derived from known facts remain unchanged even when new information is added to the knowledge base. This means that the set of derivable propositions only grows with added knowledge, never retracting conclusions. While useful in static systems, monotonic reasoning is less suited for real-time applications, where facts often change dynamically. Consequently, monotonic reasoning is common in traditional, logic-based systems, but it lacks the flexibility required for systems that need to adapt to ongoing, evolving information.

Example 2.3.2
Consider the predicates `parent(X, Y)` and `grandparent(X, Y)`, meaning respectively that `X` is parent of `Y` and that `X` is grandparent of `Y`. We also know the following facts and rules:

`parent(andrew, john).`
`parent(john, zoe).`
`grandparent(X, Y) ← parent(X, Z), parent(Z, Y).`

From facts and rules, we can infer that: `grandparent(andrew, zoe)`. Note that, if we add the new fact `parent(sarah, john)`, we get `grandparent(sarah, zoe)` that adds to the previous inference. Indeed, `grandparent(john, andrew)` still holds true because adding the new fact did not change it.

Non-monotonic reasoning works in the opposite way: adding new axioms can invalidate previously established theorems. In other words, it represents *defeasible* inference. We say that a conclusion was drawn defeasibly when we reserve the right to retract it based on new information. Defeasible inferences are common in commonsense reasoning, expert reasoning (such as medical diagnosis), and scientific reasoning. These systems are therefore crucial for modelling the beliefs of active agents, which must operate with incomplete information and update their assumptions as new information becomes available.

Example 2.3.3
Given the predicates `working_day(x)`, which stands for `x` is a working day, and `go_to_work(x)`, meaning that you are going to work on day `x`, and `ill(x)`, meaning that you are ill on day `x`, consider:

- the set of facts: $\begin{cases} \texttt{working\_day(tuesday).} \\ \texttt{working\_day(wednesday).} \end{cases}$
- the rule: `go_to_work(X)` $\leftarrow$ `working_day(X), not ill(X).`

We can infer: $\begin{cases} \texttt{go\_to\_work(tuesday)} \\ \texttt{go\_to\_work(wednesday)} \end{cases}$

Now, let us introduce a new fact: `ill(wednesday)`. Based on the updated rule, we now reconsider our plans and can just infer: `go_to_work(tuesday)`. The newly added rule causes the retraction or update of the previous conclusion about Wednesday, showing how new information can alter existing conclusions.

With definite programs, the rules we considered have excluded negative literals, as those programs are designed to represent only positive knowledge. Definite clauses, as we have seen, express that if certain premises are satisfied, specific conclusions must follow. Although they are Turing complete, there exists many cases where their expressive constraints are overly limiting. To overcome this, *general programs* have been introduced, which extend the framework to incorporate negative literals, enabling greater expressiveness and flexibility and also introducing non-monotonicity.

Definition 2.3.14 (General program)**.** *A general program is a program that permits negative literals in rule's body:*

$$H \leftarrow A_1, \ldots, A_n, not\ B_1, \ldots, not\ B_m$$

This apparently small change has many repercussions on properties of logic programs such as the fact that rules are not Horn clauses any more. Further details about this will be provided in Section 2.4. Moreover, the introduction of negation requires addressing how to manage it effectively. Indeed, if we want to use the T_P operator, defined for definite programs, we have to extend it to handle negative literals in rule bodies by requiring them to be false in the interpretation to infer the rule's head. However, this naive extension breaks monotonicity, a key requirement for the theorems that rely on the monotonic behaviour of T_P. To address this, the fixpoint semantics has to be refined, allowing them to elegantly handle negation while preserving consistency with the non-monotonic nature of such programs. There exists several operational approaches to

handling general programs. We focus on *Negation As Failure* (NAF) [37], as it is used in ASP.

2.4 Answer Set Programming

ASP is a declarative programming approach rooted in logic programming, specifically designed to model and solve combinatorial problems. It is based on the *stable model* (or *answer set*) semantics of logic programs, which represents consistent sets of logical truths that satisfy the problem constraints. In ASP, a problem is encoded as a set of logical rules, where each rule defines relationships between variables or conditions. These rules include negative literals in bodies.

The approach is particularly powerful in domains where the search space is large and where classical methods such as search or optimization algorithms may be inefficient. Its expressive power, flexibility, and ability to handle non-monotonicity make it a valuable tool in artificial intelligence and computer science. Its leading implementation, *clingo* [38], integrates the grounder *gringo* with the solver *clasp*.

2.4.1 Syntax

A common assumption in ASP is the absence of function symbols, instead ASP programs are built with predicates along with variables and constants. Predicates' name must begin with a lower case letter and usually is informative of what the predicate is about. The same happens for constants, but they can also be integer numbers. On the other hand, variables begin with an upper case letter. As in Definition 2.3.14, rules are in the form:

$$\texttt{H :- } \texttt{A}_1, \ldots, \texttt{A}_n, \texttt{ not } \texttt{B}_1, \ \ldots, \ \texttt{not } \texttt{B}_m$$

where $\texttt{H}$, $\texttt{A}_1$, ..., $\texttt{A}_n$, $\texttt{B}_1, \ldots, \texttt{B}_m$ are atoms and `:-` stands for $\leftarrow$. Notice that the operator `not` is representative for the negation as failure. Under a classical logic interpretation, this kind of rules can viewed as $\texttt{H} \vee \neg\texttt{A}_1 \vee \cdots \vee \neg\texttt{A}_n \vee \texttt{B}_1 \vee \cdots \vee \texttt{B}_m$.

ASP also allows rules without the head `H`, known as *denials*. The meaning of these rules is that the conditions in the body can never hold, as doing so would result in a contradiction.

2.4.2 Answer Set Semantics

The semantics of ASP is based on the notion of *stable model* [39] (also known as *answer set*). In the case of ASP programs without negations, we have a definite program which has a unique minimal model. As just told, the introduction of negation makes ASP non-monotonic and this unique model cannot be ensured.

Definition 2.4.1. *If P is an ASP program, then* ground(P) *is the set of all the ground instances of the rules in P.*

The grounding is performed over the Herbrand universe of the program, which, in the absence of function symbols, is finite and corresponds to the set of constants occurring

in the program. For simplicity, since the answer sets of a program P are the same of *ground(P)*, we will focus on the second one.

The answer sets of a program P belong to the minimal subsets of $\mathcal{B}_P$ which are models of *ground(P)*. To find those subsets, we need to identify a set of atoms S (which will be the answer set) such that when we transform the program P into its reduct P^S, S becomes the minimal model of P^S. This approach is effective because the reduct of P is constructed in such a way that it removes negations. By removing negations, we ensure that the program becomes simpler and the process of finding the minimal model of P^S becomes more straightforward.

Definition 2.4.2. *Given the ASP program P and S a set of atoms, the reduct of P regard S, called P^S, is the program obtained from P removing:*

1. *every rule in which body there is a negation* `not A` *such that $A \in S$,*
2. *from the remaining rules, every negative literal.*

By construction, P^S does not contain any negative literal and so it has a unique answer set. If the answer set is exactly S, then S is an answer set of P.

Example 2.4.1
Consider the program P as follows:

```
p :- not q.
```

Let us consider the following set of literals $S_1 = \{\texttt{q}\}$ and $S_2 = \{\texttt{p}\}$. To prove if they are answer sets of P we have to compute the reduct: $P^{S_1} = \emptyset$, whose answer set is $\emptyset \neq S_1$, and $P^{S_2} = \{\texttt{p.}\}$, that has a minimal model $\{p\} = S_2$ which is answer set.

We recall the following theorem, for which the proof can be found in [33].

Theorem 2.4.3 (ASP complexity)**.** *The problem of determining whether a ground general program P has stable models is NP-complete.*

Negation As Failure

As we mentioned earlier, negation in ASP follows the approach of Negation as Failure. We will now provide an intuitive explanation of this concept, which plays a central role in handling non-monotonic reasoning within the framework.

Consider a generic rule with negation (with $\texttt{n}, \texttt{m} \geq 0$):

```
p :- q, t, not r, not s.
```

In this case, `p` can be inferred if `q` and `t` can be proven and none between `r` and `s` can. This allows `p` to be inferred even when the truth value of *not* `r` and *not* `s` are unknown. As another example, consider the program:

```
p :- not q.
q :- not p.
```

In this case, the stable models are {p} and {q}, which are minimal independent models. This way of understanding negation is termed negation as failure, as we consider a literal `not p` to hold if and only if all our attempts to prove `p` fail, or we can interpret the rule as stating that one between `p` and `q` can be true while the other must be false.

2.4.3 Choice Rules and Aggregates

Choice rules are a special type of rule in ASP where syntactically the head consists of a set of literals enclosed in curly braces. These rules introduce non-determinism, allowing flexibility in the selection of which literals in the head are true. The intuitive meaning of a choice rule is: if the body of the rule is true, then none, one, or multiple literals from the head can be chosen to be true. This mechanism is particularly useful when modelling scenarios where decisions or options need to be explored or when multiple configurations are possible under certain conditions. A choice rule is written as:

$$\texttt{n}\{\texttt{L}_1;\texttt{L}_2;\ldots;\texttt{L}_k\}\texttt{m :- B}_1,\ldots,\texttt{B}_1.$$

where $\texttt{n},\texttt{m} \in \mathbb{N}$ $\texttt{L}_1,\ldots,\texttt{L}_k,\texttt{B}_1,\ldots,\texttt{B}_1$ are literals. When the body is true, a number of literals between `n` and `m` from the head are true. The set of literals can be also expressed by set-builder notation, as shown in the example below.

Example 2.4.2
Consider the following program:

```
person(zoe). person(john).
0{trust(X, Y): person(Y), X!=Y}1 :- person(X).
```

The meaning of the choice rule is that each person `X` trusts at most one person `Y`. This also includes the case where `X` trusts no one. Indeed, with two persons, we have four answer sets:

```
Answer: 1
person(zoe) person(john)
Answer: 2
person(zoe) person(john) trust(john,zoe)
Answer: 3
person(zoe) person(john) trust(zoe,john)
Answer: 4
person(zoe) person(john) trust(zoe,john) trust(john,zoe)
```

Beyond choice rules, there exists another useful extension of ASP, which comprehends aggregates. Aggregates permit to write in a compact way properties and inductive definitions involving sets of propositions. Their syntax is the following:

$$f\{t_1, \ldots, t_n : \psi\} \circ g$$

where f is an aggregation function, $t_1, \ldots, t_n (n \geq 1)$ are terms, ψ is a conjunction of literals, $\circ$ is an arithmetic comparator, and g is a ground term. Informally, it applies the aggregation function to the set of tuples $t_1, \ldots, t_n$ for which ψ holds, and then applies the comparator. Common aggregation functions are `#count`, which counts the number of element in the set of tuples, `#sum`, which sums a variable in the tuples, `#min` and `#max`, which find the minimum or the maximum of a variable in the tuples.

To conclude, the body of rules can also include *built-in atoms* of the form $E_1 \, op \, E_2$, referred to as *conditions*. These allow for arithmetic comparisons between numerical expressions, where E_1 and E_2 are composed of numerical constants, variables, and arithmetic operators, and $op \in \{<, \leq, =, !=, >, \geq\}$. During the grounding phase, variables are substituted with constants, after which the expressions are evaluated and replaced with `true` or `false`.

2.5 Summary

2.5.1 Strengths

- rules and constraints are human-readable
- Logical explanations can be checked, validated, and reasoned over
- high-level concepts can be explicitly represented
- can solve complex problems

2.5.2 Limitations

- cannot reason on continuous data

3

Inductive Logic Programming

Inductive Logic Programming [5] is a subfield of machine learning that focuses on learning logical rules from examples and background knowledge.

In this chapter, we will explore the fundamental concepts of ILP, its underlying principles, and various frameworks that have been developed to implement ILP systems. We will also discuss its integration with ASP and how ILP frameworks, such as ILASP and FastLAS, are utilized to learn from noisy and complex datasets.

3.1 Introduction

ILP is particularly powerful for tasks requiring explainability and reasoning, as it generates human-readable rules while learning from data. Indeed, unlike traditional machine learning approaches that often operate in a statistical framework, ILP exploits the symbolic representation of knowledge, using logic to construct interpretable models. ILP is particularly well-suited for learning from relational data, where examples are represented in a structured format and may involve complex interdependencies among entities. By utilizing first-order logic, ILP can derive hypotheses that are not only accurate but also interpretable, making it a valuable approach for domains requiring clear justification for decisions. Indeed, it has been widely applied in various fields, such as bioinformatics, natural language processing, and legal reasoning.

3.2 Learning Framework

Fundamentally, ILP operates on the principle of generalization: given a set of positive and negative *examples*, along with *background knowledge*, the goal is to infer a *hypothesis*, *i.e.* a set of logical rules, that accurately classifies the examples.

ILP can use different theoretical settings as a learning framework: *Learning From Entailment* (LFE), *Learning From Interpretation* (LFI), *Learning From Satisfiability* (LFS), and *Learning from Answer Set* (LAS). A particular problem associated with a framework is called a *learning task* [14] and will be denoted by T. A learning task T will

consist of a background knowledge B, which is a logic program, a hypothesis space S, defining the set of rules allowed to be in a hypothesis, and a set of examples E, whose form depends on the learning frameworks. In this section, a formal definition is given for each one, except for LAS since it is used by ILASP and so will be described more accurately in Section 3.3.2.

3.2.1 Learning From Entailment

In Learning From Entailment framework, a hypothesis is considered correct if the logical entailment of the background knowledge, combined with the hypothesis, implies the examples. In simpler terms, the learning process aims to discover rules or patterns that, when added to the existing knowledge, logically explain or predict the observed data.

Definition 3.2.1 (LFE)**.** *A* Learning from Entailment *(ILP_{LFE}) task T is a tuple $\langle B, S, E^+, E^- \rangle$ where B is a clausal theory, called the background knowledge, S is a set of clauses, called the hypothesis space and E^+ and E^- are sets of ground clauses, called the positive and negative examples, respectively. A hypothesis $H \subseteq S$ is an* inductive solution *of T if and only if:*

- $B \cup H \models E^+$,
- $B \cup H \cup E^- \not\models \bot$.

where $\models$ denotes the logical consequence and $\bot$ denotes false.

In the common case where both the background knowledge and hypothesis space consist of definite clauses, and the examples are positive or negative literals, $B \cup H$ is guaranteed to have a unique minimal Herbrand model M. H is an inductive solution if and only if M includes all positive examples and excludes (the negation of) any negative examples. In other words, $B \cup H$ must entail all positive examples and must not entail the negation of any negative examples.

Example 3.2.1
Consider the following ILP_{LFE} task $\mathrm{T} = \langle B, S_M, \langle E^+, E^- \rangle \rangle$ where:

$$\mathrm{B} = \left\{ \begin{array}{l} \texttt{parent(X, Y) :- father(X, Y).} \\ \texttt{parent(X, Y) :- mother(X, Y).} \\ \texttt{father(erik, melody).} \\ \texttt{mother(ariel, melody).} \\ \texttt{father(tritone, ariel).} \\ \texttt{mother(atena, ariel).} \end{array} \right.$$

$$\mathrm{S}_M = \left\{ \begin{array}{l} h_1 : \texttt{grandfather(X, Y):- parent(X, Z), parent(Z, Y).} \\ h_1 : \texttt{grandfather(X, Y):- father(X, Z), parent(Z, Y).} \\ h_1 : \texttt{grandfather(X, Y):- mother(X, Z), parent(Z, Y).} \end{array} \right.$$

$$\mathrm{E}^+ = \{ \texttt{grandfather(tritone, melody).}$$

$$\mathrm{E}^- = \left\{ \begin{array}{l} \texttt{:- grandfather(tritone, erik).} \\ \texttt{:- grandfather(atena, melody).} \end{array} \right.$$

Given this task, what we can conclude is that $\emptyset \notin ILP_{LFE}(T)$, as B does not entail `grandfather(tritone, melody)` (which belongs to E^+). Furthermore, any

subset of h_1, h_2, h_3 containing h_1 and/or h_3 cannot be a solution of the task because, denoting such a subset by H, we observe that $B \cup H$ entails `grandfather(atena, melody)` (which belongs to E^-), making $B \cup H \cup$ `grandfather(atena, melody)` inconsistent. The solution is h_2, as $B \cup h_2$ entails `grandfather(tritone, melody)` (which is in E^+) and is consistent with the negative examples.

3.2.2 Learning From Interpretations

The second typical framework for ILP is Learning From Interpretations.

Definition 3.2.2 (LFI)**.** *A* Learning from Interpretations *(ILP_{LFI}) task T is a tuple $\langle B, S, E^+, E^- \rangle$ where B is a definite clausal theory, S is a set of clauses and each element of E^+ and E^- is a set of facts. A hypothesis $H \subseteq S$ is an* inductive solution *of T if and only if:*

- $\forall e^+ \in E^+ : M(B \cup \{e^+\})$ *satisfies* H*,*
- $\forall e^- \in E^- : M(B \cup \{e^-\})$ *does not satisfy* H*.*

where $M(\cdot)$ is the minimal model of $(\cdot)$.

This means that, if we do not have a background knowledge, each positive example must be a model of H, and no negative example should be a model of H.

Example 3.2.2
Consider the following ILP_{LFI} task $\mathrm{T} = \langle B, S_M, \langle E^+, E^- \rangle \rangle$ where:

$$\mathrm{B} = \left\{ \texttt{daughter(X,Y):- female(X), child(X,Y).} \right.$$

$$\mathrm{S}_M = \left\{ \begin{array}{l} h_1 : \texttt{mother(X,Y):- daughter(Y,X).} \\ h_2 : \texttt{mother(X,Y):- child(Y,X).} \\ h_3 : \texttt{mother(X,Y):- child(Y,X), female(X).} \end{array} \right.$$

$$\mathrm{E}^+ = \left\{ \begin{array}{l} \texttt{child(ariel, tritone).} \\ \texttt{female(ariel).} \\ \texttt{child(melody, ariel).} \\ \texttt{female(melody). female(ariel).} \\ \texttt{mother(ariel, melody).} \end{array} \right.$$

$$\mathrm{E}^- = \left\{ \begin{array}{l} \texttt{child(melody, ariel).} \\ \texttt{female(melody). female(ariel).} \end{array} \right.$$

First of all we search for all minimal models of B combined with each examples (M_1 and M_2 from positive examples, M_3 from the negative):

- $M_1 = \{$`child(ariel, tritone), female(ariel), daughter(ariel, tritone)`$\}$
- $M_2 = \{$`child(melody, ariel), female(melody), female(ariel), mother(ariel, melody), daughter(melody, ariel)`$\}$
- $M_3 = \{$`child(melody,ariel), female(melody), female(ariel), daughter(ariel,tritone)`$\}$

Now we have to search for an hypothesis H such that M_1 and M_2 are models of H

while M_3 is not. $\emptyset$ is not a good one since M_3 is a model of $\emptyset$. Hypotheses containing h_1 are not also solutions as M_1 is not a model of h_1. The same happens for h_2. The only one which has M_1 and M_2 but not M_3 as model is h_3, so $\{h_3\} \in ILP_{LFI}(T)$.

3.2.3 Learning From Satisfiability

In the Learning From Satisfiability framework, the goal is to find a hypothesis such that, for each positive example, there exists a logical model that satisfies the hypothesis, and for each negative example, no model satisfies the hypothesis.

Definition 3.2.3 (LFS)**.** *A* Learning from Satisfiability *(ILP_{LFS}) task T is a tuple $\langle B, S, E^+, E^- \rangle$ where B is a clausal theory, S is a set of definite clauses and E^+ and E^- are sets of clausal theories. A hypothesis $H \subseteq S$ is an inductive solution of T if and only if:*

- $\forall e^+ \in E^+ : B \cup H \cup \{e^+\}$ *has at least one model,*
- $\forall e^+ \in E^- : B \cup H \cup \{e^-\}$ *has no models.*

In [40] is proved that ILP_{LFS} can simulate ILP_{LFE} using the fact that $P \models e$ if and only if $P \cup \{\neg e\}$ has no models.

Example 3.2.3
Consider the following ILP_{LFS} task $\text{T} = \langle B, S_M, \langle E^+, E^- \rangle \rangle$ where:

$$\text{B} = \left\{ \texttt{daughter:- female, child.} \right.$$

$$\text{S}_M = \left\{ h_1 : \texttt{daughter.} \right.$$

$$\text{E}^+ = \left\{ \begin{array}{l} \texttt{child.} \\ \texttt{:- female.} \end{array} \right.$$

$$\text{E}^- = \left\{ \begin{array}{l} \texttt{:- child.} \\ \texttt{:- female.} \end{array} \right.$$

The empty set is not a valid solution of the task because $B \cup \{$`:- daughter. :- female.` $\}$ is satisfiable. The solution is h_1 as $B \cup \{h_1\} \cup \{$`child.:- female.`$\}$ is satisfiable (positive example) while $B \cup \{h_1\} \cup \{$`:- daughter.:- female.`$\}$ is not (negative example).

3.3 ILASP

ILASP [14] is a logic-based machine learning system. It makes use of existing knowledge base, containing anything known before the learning starts or even previously learned rules, to infer new rules. In other words, it performs the reverse process of ASP: rather than producing solutions, it starts with examples of solutions, encodes them as training data, and works to derive an ASP program that captures the underlying rules of the real-world problem being studied.

Unlike many other ILP systems, which are only able to learn definite logic programs, ILASP is able to learn normal rules, choice rules and hard or weak constraints (that is the subset of ASP accepted by ILASP). We will use the term *rule* to refer to any of these four components.

Since we can use all those kind of rules, ILASP can be applied to:

- *preference learning*: it focuses on developing models or rules that capture and predict preferences among alternatives based on observed data. The goal is to learn how a decision-maker prioritizes options, *i.e.* it learns users' individual preferences (both from real and from synthetic data),
- learning *common-sense knowledge*, including defaults and exceptions, and
- learning *non-deterministic theories*, thanks to choice rules.

In this section, we will explain how the system works and describe its syntax.

3.3.1 Introduction

ILASP is an ILP framework and so it is used to solve learning tasks, which, as we have seen in the Section 3.2, consists of three main components: background knowledge, hypothesis space and examples.

The background knowledge is represented as an ASP program encoding known concepts or facts prior to the learning process. This serves as a foundation upon which learning is performed. ILASP operates within a subset of ASP, which varies depending on the version used. In addition to the background knowledge, which defines pre-existing rules, the user must specify the hypothesis space, *i.e.*, the set of all potential rules that ILASP is allowed to learn. This space is typically constrained using a language bias, a formal mechanism that restricts the search space to improve efficiency and relevance. ILASP supports several methods for defining such biases, including *mode declarations* and *metarules*, commonly used in other ILP systems. The four primary methods for specifying the hypothesis space in ILASP are formalized in Section 3.3.4.

The final component is the set of examples, both positive and negative, which define the semantic properties that the learned program must satisfy. An example is considered *covered* when its semantic property holds in the combined program consisting of the learned hypothesis and the background knowledge. Ideally, the learned hypothesis should cover all examples, but in real-world scenarios, this is often infeasible due to noisy data. To address this, ILASP allows assigning a penalty to each example [41], representing the cost of not covering it. The treatment of noisy examples will be discussed in greater detail in Section 3.4, as we have specifically worked with them in FastLAS.

3.3.2 Learning from Answer Set

ILASP use the ILP framework Learning from Answer Set, due to this we have to switch from classical logic to logic of *stable model* [39]. The first thing to notice is that in ASP there can be zero or many answer sets of a program. For this reason we talk about *brave entailment* ($\models_b$) and *cautious entailment* ($\models_a$): an atom a is bravely entailed by a program P if and only if at least one answer set P contains a, while it is cautiously entailed if every answer set contains it. We will denote with $AS(P)$ the set of all answer sets of P.

Definition 3.3.1. *A* cautious induction *(ILP$_c$) task T is a tuple $\langle B, S, E^+, E^- \rangle$ where B is an ASP program, S is a set of ASP rules and E^+ and E^- are sets of ground atoms.*

A hypothesis $H \subseteq S$ is an inductive solution of T if and only if $AS(B \cup H) \neq \emptyset$ and $\forall A \in AS(B \cup H), E^+ \subseteq A$ and $E^- \cap A = \emptyset$.

This means that all examples should be covered in *every* answer set and that $B \cup H$ should be satisfiable: an atom a is *cautiously* entailed by a program P iff *every* answer set contains it.

Example 3.3.1
Consider the ILP_c task $\mathrm{T} = \langle B, S, E^+, E^- \rangle$ where:

$$\mathrm{B} = \begin{cases} \texttt{dog(X):- great\_dane(X).} \\ \texttt{dog(X):- corgi(X).} \\ \texttt{great\_dane(scooby).} \\ \texttt{corgi(muick).} \end{cases}$$

$$\mathrm{S} = \begin{cases} h_1 : \texttt{long\_legs(X):- dog(X).} \\ h_2 : \texttt{long\_legs(X):- dog(X), not corgi(X).} \\ h_3 : \texttt{0\{long\_legs(X)\}1:- dog(X).} \\ h_4 : \texttt{0\{long\_legs(X)\}1:- dog(X), not corgi(X).} \end{cases}$$

$$\mathrm{E}^+ = \texttt{long\_legs(scooby)}$$
$$\mathrm{E}^- = \texttt{long\_legs(muick)}$$

The inductive solution $ILP_c(\mathrm{T})$ is h_2 because $B \cup \{h_2\}$ has exactly one answer set, and it contains `long_legs(scooby)` but not `long_legs(muick)`.

Definition 3.3.2. *A* brave induction *(ILP_b) task T is a tuple $\langle B, S, E^+, E^- \rangle$ where B is an ASP program, S is a set of ASP rules and E^+ and E^- are sets of ground atoms. A hypothesis $H \subseteq S$ is an inductive solution of T if and only if $\exists A \in AS(B \cup H)$ such that $E^+ \subseteq A$ and $E^- \cap A = \emptyset$.*

Brave induction requires all the examples to be covered by *at least one* answer set: an atom a is *bravely* entailed by a program P iff *at least one* answer set contains it.

Example 3.3.2
Consider the ILP_b task $\mathrm{T} = \langle B, S, E^+, E^- \rangle$ where B, S, E^+, E^- are the same as ones of example 3.3.1. $\{h_2\}$, $\{h_3\}$, $\{h_4\}$ are solutions of $ILP_b(\mathrm{T})$ as each of $AS(B \cup \{h_2\})$, $AS(B \cup \{h_3\})$ and $AS(B \cup \{h_4\})$ contains `long_legs(scooby)` but not `long_leg(muick)`.

The problem of brave induction is that it cannot learn constraints because it is not able to reason about what is true in *every* answer set. To overcome this, as we will see in the next section, ILASP uses *Induction of Stable Models* which allows setting conditions over multiple answer sets. The idea is to use partial interpretations as examples.

Definition 3.3.3. *A* partial interpretation e *is a pair of sets of atoms $\langle e^{inc}, e^{exc} \rangle$. We refer to e^{inc} and e^{exc} as the inclusions and exclusions respectively. An interpretation I is said to* extend *e if and only if $e^{inc} \subseteq I$ and $e^{exc} \cap I = \emptyset$.*

ILASP learns through brave induction from positive examples, while negative are cautiously entailed. This means that there is no guarantee that e^{exc} will always be excluded.

Definition 3.3.4 (Induction of Stable Models)**.** *An* induction of stable models *(ILP_{sm}) task T is a tuple $\langle B, S, E\rangle$, where B is an ASP program, S is the hypothesis space and E is a set of example partial interpretations. A hypothesis H is an inductive solution of T if and only if $H \subseteq S$ and $\forall e \in E, \exists A \in AS(B \cup H)$ such that A extends e.*

Keeping in mind all these definitions, we can finally declare what is Learning from Answer Sets.

Definition 3.3.5. *A* Learning from Answer Sets *task is a tuple $T = \langle B, S, E^+, E^-\rangle$ where B is an ASP program, S a set of ASP rules and E^+ and E^- are finite sets of partial interpretations. A hypothesis $H \subseteq S$ is an inductive solution of T if and only if:*

- $\forall e^+ \in E^+ \ \exists A \in AS(B \cup H)$ *such that A extends e^+,*
- $\forall e^- \in E^- \ \nexists A \in AS(B \cup H)$ *such that A extends e^-.*

3.3.3 Examples

ILASP examples are partial interpretations. As shown above, positive examples have to be extended by at least one answer set, so they can be considered as characterised by brave induction, while negative examples as characterised by cautious induction, because we want them to not be extended by any answer set. Traditionally, ILP frameworks have operated on the premise that hypotheses should cover all the examples based on a single predetermined set of background knowledge. However, in real-world scenarios, certain examples may exhibit context-dependent characteristics, necessitating the use of distinct background knowledge for covering different examples. This is accomplished by the use of a *Context-Dependent Partial Interpretation* (CDPI) as examples. Imagine you are trying to learn the rules of a board game, but the board can have varying dimensions. In this case, the specific dimensions should be included in the examples rather than in the background knowledge, as they are instance-specific and not part of the general rules. Allowing to do so permits ILASP to learn rules that are sensitive to varying scenarios.

Definition 3.3.6. *A context-dependent partial interpretation is a pair $e = \langle e_{pi}, e_{ctx}\rangle$, where e_{pi} is a partial interpretation and e_{ctx} is an ASP program, called a context. Given a program P, an interpretation I is said to be an accepting answer set of e wrt P if and only if $I \in AS(P \cup e_{ctx})$ and I extends e_{pi}. P is said to accept e if there is at least one accepting answer set of e wrt P.*

3.3.4 Hypothesis Space

As already said, rules that can be learnt must belong to the hypothesis space. The construction of this set of rules can be done in four different ways:

1. The easiest is by explicitly listing all the rules we are interested in. This is the simplest method for generating the hypothesis space, but it presupposes prior knowledge of what one intends to learn. This cannot always be valid, as the aim is to automatically induce rules from generic examples.

2. The second way to generate rules is through mode declarations. These declarations use placeholders, *i.e.* terms such as `var(t)` for variables or `const(t)` for constants which are then replaced by appropriate variables or constants of the specified type `t`. Constants allowed to replace a `const(t)` placeholder are defined with the predicate `#constant(t, c)`. Those placeholders are used as arguments in mode declarations, as they are ground atoms. An atom is compatible with a mode declaration if each placeholder is correctly replaced by a corresponding variable or constant. ILASP ensures that no variable appears twice in the same rule with a different type.

   There are four types of mode declarations in ILASP: `#modeh` (head declarations), `#modeha` (aggregate head declarations), `#modeb` (body declarations), `#modec` (condition declarations), `#modeo` (optimization body declarations). Each mode declaration can include a recall parameter, which limits the number of times a particular declaration can be used in a rule (*e.g.*, `#modeb(2, p(var(t)))` allows predicate `p` to appear twice in a rule with a variable of type `t`). Furthermore, constraints such as `#maxv` (maximum number of variables per rule) and `#max_penalty` (upper bound on hypothesis size) can be specified.

3. The other two techniques to construct the hypothesis space are the *meta-level ASP definition* and the *metarules*. We will just cite them since they are not used in this thesis.

Once the hypothesis space is defined, ILASP searches for the optimal hypotheses inside such space. This optimality is defined in terms of the number of literals in the hypothesis, which will be the measure of the complexity of the hypotheses space, also called *lenght*. In cases where the rule also contains an aggregate, the computation of the length is slightly different. Indeed, counting just the occurrences of literals does not work well for aggregates. To solve this, aggregates are converted into *disjunctive normal form* (DNF), *i.e.* a disjunction of conjunctions of literals, considering both the number of answer sets and literals involved.

Example 3.3.3
Consider the aggregates `1{a, b}1` and `1{a, b}2`: if the length consists of the number of literals then they would have the same length (2) even if their semantics is different. Instead, if we convert them in DNF we get:
(`a` ∧ `not b`) ∨ (`not a` ∧ `b`), for the first one, and
(`a` ∧ `b`) ∨ (`a` ∧ `not b`) ∨ (`not a` ∧ `b`), for the second one.
Now the length for the first is 4, while for the second is 6.

We can now define formally what is the length of an hypothesis.

Definition 3.3.7. *Given a hypothesis H, the length of the hypothesis, $|H|$, is the occurrences of literals that appear in H^D, where H^D is constructed from H by converting all aggregates in H to disjunctive normal form.*

3.3.5 ILASP Syntax

The syntax for background knowledge in ILASP is similar to ASP, except that defining constants is not permitted. Positive (E^+) and negative (E^-) examples are declared as follows: #pos({e^{inc}}, {e^{exc}}) for positive examples (resp. #neg for negative ones).

To describe hypothesis spaces, as described in sect. 3.3.4, there are several approaches, we will see the two that have been used in this thesis:

1. mode declarations: A mode declaration is a ground atom with placeholders like var(t) or const(t), where var(t) and const(t) can be replaced by any variable or constant of type t. The syntax is:
   - #modeh(pred(·)) for declaring head predicates, or #modeha(pred(·)) if aggregates are allowed.
   - #modeb(n, pred(·)) for body predicates, where n specifies the maximum occurrences of pred(·) in the body.
   - #modec(·) for conditions, e.g., $\texttt{var}(t_1) > \texttt{var}(t_2)$.
2. direct rule declarations: rules can be explicitly declared to define the hypothesis space. In this case each rule in the list must be preceded by the length of the rule (*e.g.* 3 ∼ `p:- r, not s.`).

3.3.6 Complete Example of ILASP Encoding

We present a complete ILASP encoding for modelling the *n-queens* puzzle. This puzzle involves placing n queens on an $n \times n$ chessboard such that no two queens attack each other, which means no two queens can share the same row, column, or diagonal. The predicate `queen(X, Y)` is used to specify that a queen is located at position `(X,Y)` on the board. The background knowledge will consist in just three rules that describe the not allowed positions of queens on the board. In particular $\texttt{same}_\texttt{c}\texttt{ol}(\texttt{X1}, \texttt{Y1}, \texttt{X2}, \texttt{Y2})$ means that there are two queens on the board, respectively in (X1, Y1) and in (X2, Y2), and that those two positions are on the same column. Similar for $\texttt{same}_\texttt{r}\texttt{ow}(\texttt{X1}, \texttt{Y1}, \texttt{X2}, \texttt{Y2})$ and $\texttt{same}_\texttt{d}\texttt{iagonal}(\texttt{X1}, \texttt{Y1}, \texttt{X2}, \texttt{Y2})$.

The hypothesis space is here described with direct rule declarations. In particular, the rules in it are denials for not allowed queens' positions and choices rules to ensure that exactly n queens are placed on the board.

For ILASP examples encoding, it was sufficient to consider one positive and one negative example for each constraint we wanted to learn.

```
%% Background knowledge
same_col(X1,Y,X2,Y) :- queen(X1,Y), queen(X2,Y), X1!=X2.
same_row(X,Y1,X,Y2) :- queen(X,Y1), queen(X,Y2), Y1!=Y2.
same_diagonal(X1,Y1,X2,Y2) :- queen(X1,Y1), queen(X2,Y2), X1!=X2,
                               |X1-X2|=|Y1-Y2|.

%% Hypothesis space
1 ~ :- same_row(X1, Y1, X2, Y2).          % only one queen per row
1 ~ :- same_col(X1, Y1, X2, Y2).          % only one queen per column
1 ~ :- same_diagonal(X1, Y1, X2, Y2).     % only one queen per diagonal
```

```
2 ~ 1{queen(X,Y): coord(Y)}1 :- coord(X). % only one queen per row
2 ~ 1{queen(X,Y): coord(X)}1 :- coord(Y). % only one queen per column
4 ~ 4{queen(X,Y): coord(X), coord(Y)}4.   % four queens in total


%% Examples
%% in every column there cannot be more than one queen
#pos({queen(1,3)}, {queen(2,3), queen(3,3), queen(4,3)}).
#neg({queen(4,3), queen(2,3)},{}).
%% in every row there cannot be more than one queen
#pos({queen(1,3)}, {queen(1,2), queen(1,4), queen(1,1)}).
#neg({queen(2,1), queen(2,4)},{}).
%% in every diagonal there cannot be more than one queen
#pos({queen(2,4)}, {queen(3,3), queen(4,2), queen(1,3)}).
#neg({queen(1,1), queen(4,4)},{}).
```

Using this set of examples and hypothesis space, the program successfully learns the second and third constraints as well as the first choice rule. Note that, it learns the choice rule instead of the first constraint because the rule is essential for generating answer sets containing instances of `queen(X, Y)`. Other constraints from S are discarded, even if valid, because ILASP prioritizes returning the *minimal* set of rules needed to satisfy the examples.

3.3.7 Algorithm

There exist different versions of ILASP algorithm, but all of them, like many ILP systems, adhere to the principle of *Occam's Razor*, favouring the "simplest" solution that satisfies the given examples. Occam's Razor is, indeed, a philosophical principle that suggests that, among competing hypotheses that explain the same set of observations, the simplest one should be preferred. In practice, this involves searching for an optimal hypothesis. In the context of ILP, this principle is operationalised by favouring hypotheses with minimal complexity, typically measured in terms of the number or length of logical rules. The goal is to induce a hypothesis that not only explains the training data (*i.e.*, covers all positive examples and excludes negative ones) but does so with as few rules, literals, or predicates as necessary. Furthermore, simpler hypotheses are generally more interpretable and tend to generalise better to unseen data, reducing the risk of overfitting. However, strict adherence to simplicity can sometimes limit expressiveness. More recent systems, such as FastLAS (see Section 3.4.3), expand on this principle by allowing custom scoring functions that balance simplicity with other domain-specific preferences or constraints.

The first two versions, ILASP1 and ILASP2, follow the same three phases:

1. *pre-processing*, when they map the input learning task into an ASP program
2. *solving*, when the ASP program produced from the previous step is solved by Clingo solver,

3. *post-processing*, when the answer set returned by the solver is post-processed to extract the learned program.

During the solving phase, both versions employ a *multi-shot* technique but in a different way. The goal is to compute two types of programs, known as *positive hypotheses* and *violating hypotheses*. The first type includes programs that cover all positive examples, ensuring they are fully satisfied. The second type includes programs that also cover all positive examples but fail to cover at least one negative example, helping to identify potential inconsistencies in the learned rules. This approach allows the program to be solved iteratively, with new components added or existing ones removed at each iteration. Starting at $n = 0$, the ILASP1 algorithm performs the following steps:

1. compute all violating hypotheses of length n,
2. convert each violating hypothesis into a constraint,
3. add the computed constraints to the program,
4. search for a positive hypothesis of length n that satisfies all constraints, ensuring it covers all positive examples,
5. if the previous step succeeds then the program is returned, otherwise n is incremented and a new iteration starts.

ILASP2 optimises the process by computing a single optimal positive solution in each iteration, improving efficiency. This approach addresses the issue of handling a potentially large number of violating hypotheses, which, in ILASP1, could result in as many constraints being added to the program. ILASP2 steps are:

1. compute a single optimal positive solution,
2. if it is a violating hypothesis, identify a *violating reason* (*vr*) explaining why the hypothesis is invalid,
3. encode *vr* into ASP rules and add them to the program (this ensures the current hypothesis and any other programs violating for the same reason *vr* are excluded in subsequent iterations)
4. continue until the solution is not violating.

The advantage of ILASP2 is that each iteration adds fewer elements to the program compared to ILASP1. This efficiency is further enhanced by the use of violating reasons, which target and eliminate multiple violating hypotheses simultaneously. As a result, ILASP2 requires fewer iterations to produce a solution. In the Fig. 3.1 the behaviour of ILASP 1 and 2 are visually explained.

The main limitation of the ASP encoding used in ILASP1 and ILASP2 is that its grounding scales with the number of examples in the learning task, making it inefficient for large datasets. To address this, it is beneficial to reduce the number of examples without losing essential information. In real datasets, many examples overlap in the concepts they represent, allowing examples to be grouped into classes based on the programs that cover them. In noise-free settings, only one example per class is necessary,

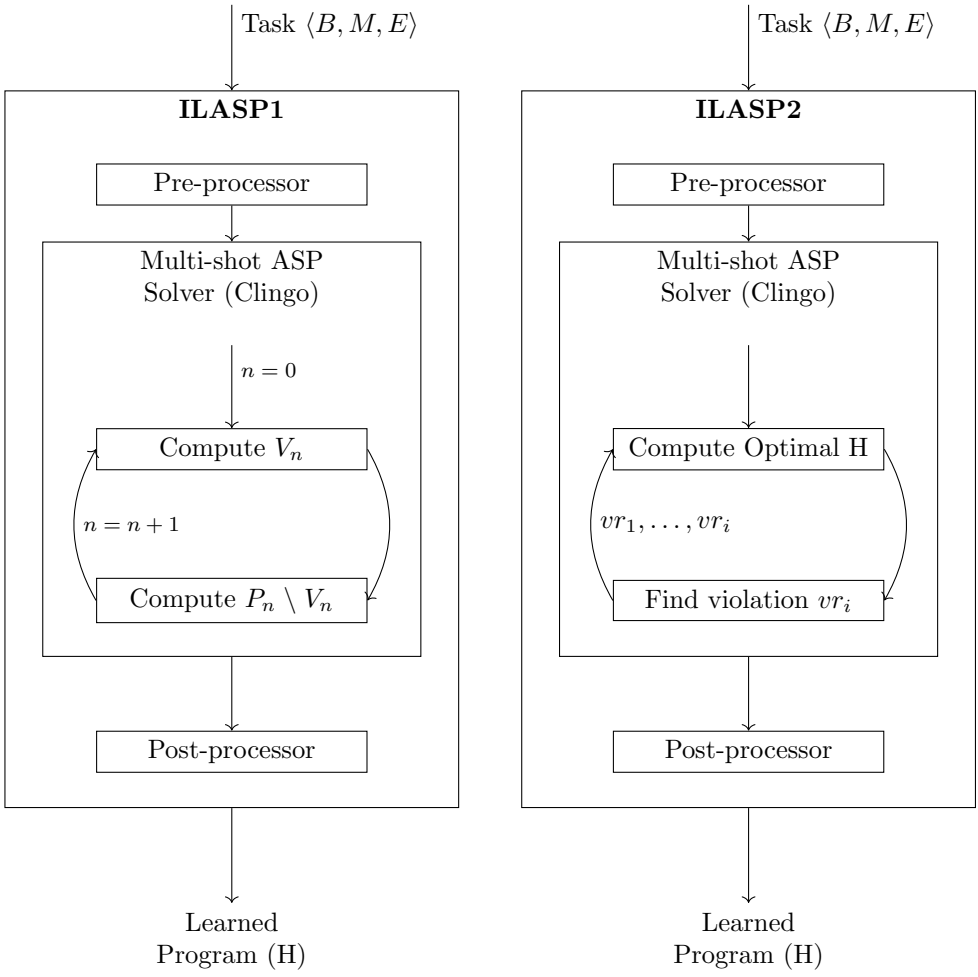


Figure 3.1: Comparison between ILASP1 (left) and ILASP2 (right), where V_n corresponds to violating hypotheses of length n and P_n to positive once, and $vr_1, \ldots, vr_i$ are the violating reasons.

as the others become redundant. ILASP2i exploits this by constructing a smaller subset of examples, known as *relevant examples*, which importantly reduces the dataset size while preserving all necessary information. The construction of relevant examples in ILASP2i proceeds as follows (also shown in Fig. 3.2), starting with a program H initially set to empty, as it is the shortest program that covers the (empty) set of relevant examples:

1. search for a new relevant example not covered by the current program H,
2. if a relevant example is found, update H by searching for a new program that covers all identified relevant examples (using ILASP2),

3. if no relevant example exists, H is considered an optimal solution and returned.

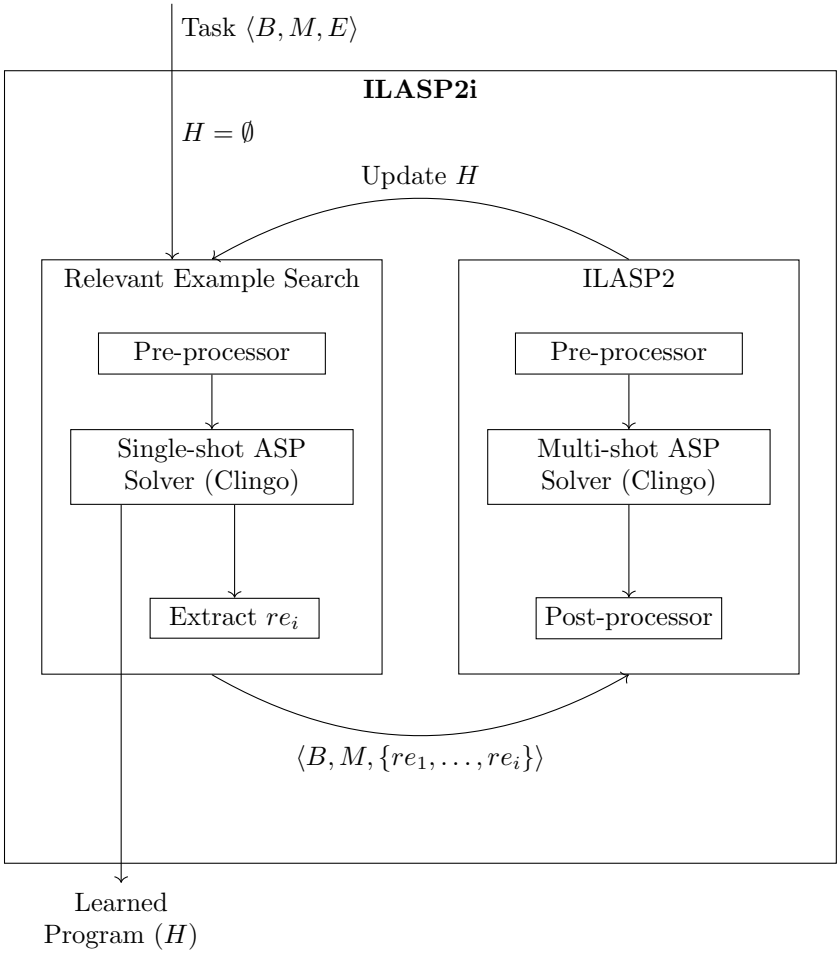


Figure 3.2: ILASP2i, where $\{re_1, \ldots, re_i\}$ are the relevant examples.

Unfortunately, as noted, this approach is effective only in noise-free examples. However, when applied to noisy datasets, the penalty of a class of examples is the sum of the penalties of the examples that were found to be relevant, that is then evaluated to check if the class is worthwhile covering. This process can slow down the computation, as multiple relevant examples from the same class may need to be found before their cumulative penalty becomes significant enough to justify coverage. In some cases, this can result in ILASP2i being slower than ILASP2 [42].

The solution is provided by ILASP3, which utilises *coverage constraints*. These constraints are generated by a dedicated module called the *Example Translator*, which, as the name suggests, converts an example into a set of coverage constraints. These constraints specify conditions satisfied by exactly the programs that cover the translated

example. The Example Translator module not only translates relevant examples into a set of constraints but also performs the *implication check* step to determine which other examples require those constraints as necessary conditions for coverage. These coverage constraints approximate both the coverage and score of every program within the program space. Importantly, this approximation of a program's score is always guaranteed to be less than or equal to the program's real score, ensuring a conservative yet reliable evaluation of potential solutions. This approach enables ILASP3 to use a single-shot ASP call instead of a multi-shot one. Once constraints for one example are generated, they can be reused to check if they are necessary for covering other examples. This methodology significantly reduces the computational cost of the optimal program search and, more importantly, it enhances the system's ability to handle noisy tasks effectively. Indeed, once a relevant example is found and translated, the computed constraints must be satisfied by the learned program, otherwise, an entire set of examples remains uncovered, and the sum of penalties for all these examples must be incurred. This differs from ILASP2i, where only the single relevant example's penalty is considered, therefore ILASP3 often requires fewer iterations than ILASP2i.

ILASP4 introduces important improvements over its predecessor, ILASP3, particularly in how it handles conflict analysis and constraint generation. While ILASP3 follows a conflict-driven approach [43], where relevant examples (conflicts) are directly translated into constraints on the solution space, its method of constraint generation can be computationally expensive. This is because ILASP3 ensures that constraints are both necessary and sufficient for an example to be covered, leading to long and complex constraints that can take considerable time to compute.

In contrast, ILASP4 adopts a more relaxed approach by generating only necessary constraints, which may not always be sufficient. This trade-off results in a potentially higher number of iterations during the learning process, as the same example might be reconsidered multiple times. However, since each iteration requires significantly less computation, ILASP4 is generally faster in practice. This distinction becomes particularly important in non-categorical learning tasks, where programs may have multiple answer sets. In such cases, ILASP3's tendency to generate long constraints can be a bottleneck, whereas ILASP4's shorter constraints improve efficiency.

Another key improvement in ILASP4 is the modularity of its *Conflict-Driven Inductive Logic Programming* (CDILP) framework. Indeed, ILASP4 allows users to customize components of the conflict analysis and constraint propagation mechanisms, enabling domain-specific optimizations. This flexibility is made easier through a Python-based interface, PyLASP. As long as user-defined modifications maintain the correctness properties of the original modules, the system guarantees termination and the discovery of an optimal solution.

Overall, while ILASP4 may require more iterations than ILASP3, the computational savings per iteration make it more practical for complex learning tasks. Its modular framework further enhances its applicability, allowing to fine-tune the learning process to suit specific needs.

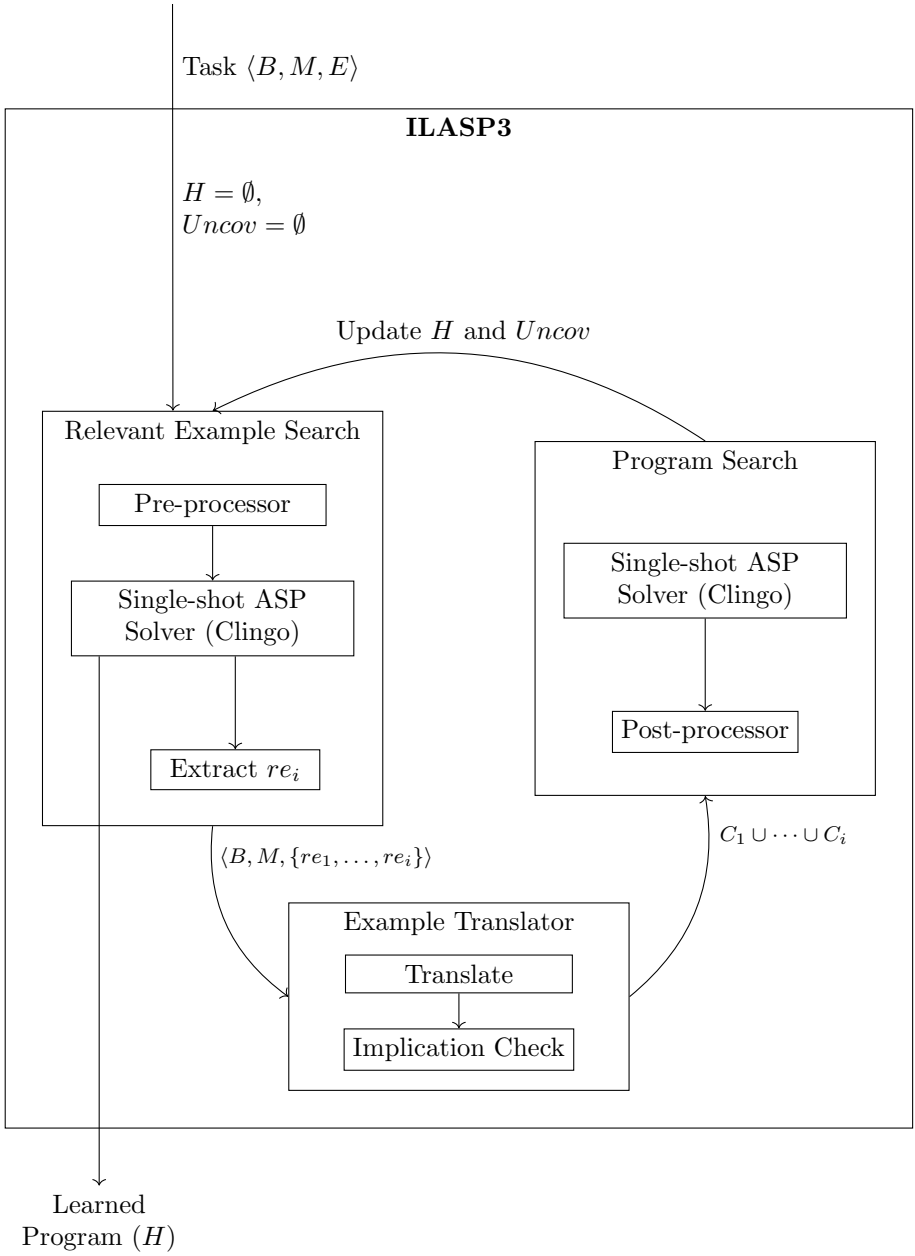


Figure 3.3: ILASP3, where $\{C_1, \ldots, C_i\}$ are the coverage constraints e $Uncov$ is the set of examples which are known not to be covered by the hypothesis (according to the coverage constraints).

3.4 FastLAS

In scenarios with multiple hypotheses, ILP systems often prioritise shorter solutions, leading to overly general rules. However, in real-world applications like access control policies, overly generic rules may not be ideal. FastLAS [15] offers a solution by exploiting Inductive Logic Programming to learn more specific rules. Built on the Context-dependent Learning from Answer Sets framework of ILASP, FastLAS enhances scalability and introduces scoring functions over hypotheses, enabling users to define their own optimization criteria and tailor the learned rules to specific needs.

In the following section we are going to describe how this framework works.

3.4.1 Introduction

The idea behind FastLAS is similar to the one of ILASP but offers other techniques that make it also faster and more scalable. As ILASP, it takes as input not only a learning task but also a customised scoring function so the solution computed is optimal with regards to the given scoring function. Moreover, FastLAS' learning algorithm introduces an innovative method by identifying a reduced subset of the hypothesis space, known as an *OPT-sufficient* subset, which is guaranteed to contain at least one optimal solution according to the specified scoring function. By focusing on this smaller search space, which can be notably smaller than the entire hypothesis space for a given learning task, FastLAS ensures efficiency. As said, the algorithm guarantees the return of an optimal solution based on the scoring function: among multiple hypotheses that explain the examples, the hypothesis with the lowest score is selected.

3.4.2 Observational Predicate Learning from Answer Sets

The first version of FastLAS was focused on solving a specific subset of context-dependent LAS tasks [44]. These tasks are restricted to *Observational Predicate Learning* (OPL), simplifying the problem while maintaining the ability to derive meaningful and context-aware logical rules. Working only with OPL means that the learned hypothesis defines predicates that are directly observed in the examples: it cannot learn knowledge that is indirectly observable, such as the causes of observed events. The set of a ILP^{OPL}_{LAS} tasks will be denoted by $\mathcal{T}_{OPL}$. Only in latter versions FastLAS was able to handle also *non*-OPL (*non*-Observational Predicate Learning) tasks [45].

Definition 3.4.1 (ILP^{OPL}_{LAS})**.** *An Observational Predicate Learning from Answer Sets (ILP^{OPL}_{LAS}) task is a tuple $T = \langle B, S_M, E^+ \rangle$ where B is an ASP program called background knowledge, S is a hypothesis space and M is a mode bias, and E^+ is a set of* Weighted Context-Dependent Partial Interpretation *(WCDPI)s such that $\forall e \in E^+, |AS(B \cup e_{ctx})| = 1$, and no predicate in M_h occurs in M_b or in the body of any rule in T (where M_h is mode bias for the heads of the rules and M_b for the body). We will say that:*

- *a hypothesis $H \subseteq S_M$ covers an example $e \in E^+$ iff $B \cup H$ accepts e,*
- *H is an inductive solution of T (written $H \in ILP^{OPL}_{LAS}(T)$) iff $\forall e \in E^+$ s.t. $e_{pen} = \infty$, H covers e,*
- *T is satisfiable iff $ILP^{OPL}_{LAS}(T) \neq \emptyset$.*

3.4.3 Customizable Scoring Function

The new aspect of FastLAS lies in its use of a scoring function to distinguish between multiple inductive solutions for a given task, enabling the selection of the optimal hypothesis based on user-defined criteria. While all ILP systems employ some form of cost function to guide the search for the best hypothesis, they differ in how this cost is defined. For instance, ILASP relies primarily on hypothesis length as a measure of simplicity, following the principle of Occam's razor (see Section 3.3.7). In contrast, FastLAS extends this approach by allowing users to define custom cost components, such as penalties on specific predicates or rule structures, thus offering greater control and expressiveness in shaping the learned model.

Definition 3.4.2. *A scoring function is a function* $S : Programs \times \mathcal{T}_{OPL} \to \mathbb{R}_{\geq 0}$. *A hypothesis* $H \in Programs$ *is said to be an optimal solution of a task* $T \in \mathcal{T}_{\mathcal{OPL}}$ *w.r.t. a scoring function* S *if and only if* $H \in ILP^{OPL}_{LAS}(T)$ *and there is no* $H' \in ILP^{OPL}_{LAS}(T)$ *such that* $S(H', T) < S(H, T)$.

This approach allows fine-grained control over the costs associated with specific predicates, enabling tailored rule learning. In practice, we can assign specific penalties to predicates based on our preferences.

Example 3.4.1
Consider the following definition of a bias function:

```
#bias("penalty(3, head(X)) :- in_head(X).")
#bias("penalty(10, body(X)) :- in_body(X), X = p(_).").
```

This definition implies the following:

- every rule incurs a penalty of 3 whenever it has a head (essentially, for each rule);
- if the body of the rule also contains the predicate `p`, an additional penalty of 10 is applied.

FastLAS currently supports *decomposable* scoring functions. These can be evaluated on each rule independently. We give now its definition since it will be useful in the next section.

Definition 3.4.3 (Decomposition)**.** *Given the scoring function* S, *a function* $S^{rule} : Rules \times \mathcal{T}_{OPL} \to \mathbb{R}_{\leq 0}$ *is a decomposition of* S *iff for each* $P \in Programs$ *and* $T \in \mathcal{T}_{OPL}, S(P, T) = \sum_{r \in P} S^{rule}(r, T)$.

In Section 3.3, we mentioned that examples can also be noisy due to the nature of real-world scenarios. FastLAS, as ILASP, is able to handle this issue giving a *penalty*, or a *weight*, to the example. To do so, it has been introduced the notion of WCDPI. A WCDPI is a tuple $e = \langle e_{id}, e_{pen}, e_{pi}, e_{ctx} \rangle$, where e_{id} is an identifier for e, e_{pen} is the penalty, which is a positive integer or ∞, e_{pi} is a partial interpretation and e_{ctx} is the context. We will say that a WCDPI e is accepted by a program P if and only if there is an answer set of $P \cup e_{ctx}$ that extends e_{pi}.

3.4.4 Non-Observational Predicate Learning from Answer Set

FastLAS, as mentioned, was initially designed for Observational Predicate Learning only, where the predicates to be learned appear explicitly in the examples. However, many tasks require learning knowledge that is not directly observable, such as inferring hidden relationships or unobserved causes. These are classified as *non*-OPL tasks and pose unique challenges, particularly in the context of non-monotonic reasoning frameworks like ASP.

To address this, a novel version of FastLAS was introduced for handling non-OPL tasks. This method involves translating examples from a non-OPL task into a set of structured examples, referred to as *possibilities*. Each possibility represents a hypothetical scenario that could account for the original non-observational example. The method ensures that the original example is considered covered by the hypothesis if and only if at least one of its associated possibilities is covered. This enables an ILP system originally designed for OPL tasks to effectively solve non-OPL problems, essentially upgrading the system to handle more complex learning tasks.

The possibility-based approach achieves this by:

- constructing a space of possibilities for each non-OPL example. This step involves generating logical scenarios that align with the original example's constraints.
- ensuring that covering one possibility is sufficient to satisfy the original non-observational example. This approach aligns non-OPL tasks with the OPL framework, making them manageable for systems like FastLAS.

This advancement allows ILP systems, such as FastLAS, to handle non-OPL tasks while preserving scalability and computational efficiency. Comparative evaluations have demonstrated that the upgraded system, known as *FastNonOPL*, outperforms other ASP-based ILP systems in solving non-OPL tasks in terms of both speed and accuracy.

3.4.5 Algorithm

FastLAS algorithm guarantees to return an optimal solution of any ILP^{OPL}_{LAS} task with respect to any scoring function. It is composed of four steps: *initial construction*, *generalisation*, *optimisation*, and *solving*.

ILASP begins by computing the hypothesis space S_M, which contains every rule that is compatible with the mode bias M. FastLAS instead begins by computing an *OPT-sufficient* subset of the hypothesis space, which is often significantly smaller than S_M. Indeed, during the first step, the algorithm constructs a subset S^1_M of the hypothesis space that guarantees to contain at least one solution to the task, if the task is satisfiable. In other words, it computes a *SAT-sufficient* subset of S_M since a subset of the hypothesis space of a task is SAT-sufficient if and only if it either contains at least one solution of the task or the original task is unsatisfiable. Therefore, usually this subset consists of highly specific rules, each designed to cover only a single example, ensuring a minimal yet sufficient representation of potential solutions. Note that FastLAS is restricted to solving *non-recursive* predicate learning tasks. A task is non-recursive if no predicate in M_h occurs in M_b. In this phase it is constructed the set of *characteristic* rules. In fact, in this step, the algorithm computes, in parallel, the *characterisations* (def. 3.4.6) of each example since they are independent from each other.

Definition 3.4.4 (Characteristic ruleset)**.** *Let $e \in E^+$ and a be a ground atom. A rule R is in the characteristic ruleset of T with regard to a and e if and only if:*

- $R \in S_M$,
- *there is at least one ground instance R^g of R such that $a = head(R^g)$ and $body(R^g)$ is satisfied by the unique answer set of $B \cup e_{ctx}$,*
- *there is no rule R' that satisfies the previous two points such that R is a strict subrule of R'.*

We will denote such ruleset with $\mathcal{C}(T, a, e)$.

The characteristic ruleset can verify whether a hypothesis, combined with the background knowledge, correctly accepts a given example. Indeed, the following theorem holds.

Theorem 3.4.5. *Given an example $e \in E^+$ and $H \subseteq S_M$, we have that $B \cup H$ accepts e if and only if for each $a \in e^{inc}$ there is at least one rule in H that is a subrule of a rule in $\mathcal{C}(T, a, e)$.*

The characterisation of examples in T, referred to as $\mathcal{C}(T)$, can be used to derive a SAT-sufficient subset of a hypothesis space.

Definition 3.4.6 (Characterisation)**.** *The characterisation of an example $e \in E^+$, written $\mathcal{C}(T, e)$, is the pair $\langle C^+, C^- \rangle$, where $C^+ = \bigcup_{a \in e^{inc}} \mathcal{C}(T, a, e)$ and $C^- = \bigcup_{a \in e^{exc}} \mathcal{C}(T, a, e)$.*

Theorem 3.4.7. *Given $\mathcal{C}(T) = \{R | e \in E^+, R \in e_I\}$, T is satisfiable if and only if $\mathcal{C}(T)$ contains an inductive solution of T.*

The idea is that the union of all C^+ sets include any maximal rule that can prove at least one inclusion, ensuring it includes any rule potentially useful for covering an example. In contrast, C^- sets consist of maximal rules proving at least one exclusion and so will not end in any inductive solution. That is the reason why they are not considered important when constructing a SAT-sufficient subset.

Example 3.4.2

Consider the task T with the following background knowledge and mode bias:

$B = \emptyset$

$M_h = \{\texttt{p}, \texttt{q}\}$ $\qquad M_b = \left\{ \begin{array}{l} \texttt{r, not r,} \\ \texttt{s, not s} \end{array} \right.$

$e_1 = \langle id_1, \infty, \langle \{\texttt{p}\}, \{\texttt{q}\} \rangle, \{\texttt{r.}\} \rangle$
$e_2 = \langle id_2, \infty, \langle \{\texttt{q}\}, \{\texttt{p, s}\} \rangle, \{\} \rangle$
$e_3 = \langle id_3, \infty, \langle \{\texttt{q}\}, \{\} \rangle, \{\texttt{s.}\} \rangle$

The correspondent characterisation are computed below:

$$\mathcal{C}^+(T) = \left\{ \begin{array}{l} \texttt{q:- not r, not s.} \\ \texttt{q:- s, not r.} \\ \texttt{p:- r, not s.} \end{array} \right. \qquad \mathcal{C}^-(T) = \left\{ \begin{array}{l} \texttt{p:- not r, not s.} \\ \texttt{q:- r, not s.} \end{array} \right.$$

The next step, the generalisation, consists in searching for rules that are subrules of one or more rules in S_M^1. It will compute the generalised characteristic hypothesis space of T from $\mathcal{C}(T)$ and will end in a wider hypothesis space S_M^2.

Definition 3.4.8 (Generalised Characteristic Hypothesis Space)**.** *For any rule $R \in S_M$, let c_R be the set of all rules R' in $\mathcal{C}(T)$ such that R is a subrule of R'. The generalised characteristic hypothesis space of T, written $\mathcal{G}(T)$, is the set containing every rule R for which $c_R \neq \emptyset$ and there is no rule $R' \in S_M$ such that R is a strict subrule of R' and $c_R = c_{R'}$.*

In other words, $\mathcal{G}(T)$ contains the maximal subrules of rules in $C^+(T)$ that are subrules of as many rules in $C^+(T)$ as possible.

Example 3.4.3
Consider again Example 3.4.2. The generalization produce the following:

$$\mathcal{G}(T) = \begin{cases} \texttt{q:- not r, not s.} \\ \texttt{q:- s, not r.} \\ \texttt{p:- r, not s.} \\ \texttt{q:- not r.} \end{cases}$$

After generalisation there is optimisation, during which, each rule R in S_M^2 is modified to compute an optimal subrule R' based on the scoring function and ensuring consistency with the exclusions across all examples. To do so, FastLAS generates an *optimised characteristic hypothesis space*.

Definition 3.4.9 (Optimised Characteristic Hypothesis Space)**.** *Given $R \in S_M$ and S a decomposable scoring function, R' is an optimisation of R iff:*

- *R' is a subrule of R,*
- *$\nexists e \in E^+$ s.t. $e_{pen} = \infty$ and R' is a subrule of a rule in e_V,*
- *there is no R'' satisfying the previous points s.t. $S^{rule}(R'', T) < S^{rule}(R', T)$.*

R is optimisable if and only if it has at least one optimisation. An optimised characteristic hypothesis space $\mathcal{O}$ of T w.r.t. S is a set of rules containing at least one optimisation of each optimisable rule in $\mathcal{G}(T)$.

This means that for each $R \in \mathcal{G}(T)$, the algorithm compute one optimal subrule of R w.r.t. $\mathcal{S}$ that is not a subrule of any rule in $C^-(T)$. The hypothesis space S_M^3 that results from this phase is guaranteed to include an optimal inductive solution to the task, provided the task is satisfiable. In particular, the OPT-sufficient subset of the hypothesis space is computed during this phase.

Theorem 3.4.10. *Let $\mathcal{O}$ be an optimised characteristic hypothesis space of T w.r.t. to a decomposable scoring function $\mathcal{S}$. If T is satisfiable, then $\mathcal{O}$ contains at least one optimal inductive solution of T w.r.t. $\mathcal{S}$.*

The last phase is the solving one. At this point, FastLAS searches for the optimal inductive solution. The resolution is similar to the one done by ILASP since the learning task of FastLAS is a simplification of the one of ILASP. FastLAS is sound and complete w.r.t. the optimal inductive solution of any ILP^{OPL}_{LAS} task under any decomposable scoring function.

Example 3.4.4
Let us now compute $\mathcal{O}(T)$ for the Example 3.4.2.
We have to optimise using $\mathcal{S}$, which gives a penalty of one to each head and one to each predicate in body:

```
q:- not r, not s.          q:- not r.
q:- s, not r.    optimization  q:- not r.
p:- r, not s.    ------------> p:- r.
q:- not r.                 q:- not r.
```

Therefore the result is:

$$\mathcal{O}(T,\mathcal{S}) = \left\{ \begin{array}{l} \texttt{q:- not r.} \\ \texttt{p:- r.} \end{array} \right.$$

3.5 Summary

3.5.1 Strengths

- rules and constraints are human-readable
- Logical explanations can be checked, validated, and reasoned over
- high-level concepts can be explicitly represented

3.5.2 Limitations

- cannot predict continuous data
- problems with scalability
- low accuracy

4

Explainable AI

Artificial Intelligence systems, particularly those based on deep learning, have demonstrated remarkable performance across a wide range of domains. However, despite their effectiveness, many of these models operate as *black-box* (BB), providing little to no insight into the reasoning behind their predictions. This lack of transparency poses considerable challenges in critical applications where interpretability is essential, such as healthcare, finance, and legal decision-making. Consequently, the field of Explainable AI has emerged to address these concerns by developing methods that enhance the interpretability and trustworthiness of AI systems.

This chapter examines the various alternatives of explainability in AI, with a focus on logic-based approaches for enhancing the interpretability of deep learning models. We discuss the role of symbolic reasoning in explaining AI predictions, the integration of ILP techniques, and the challenges associated with making complex AI models more transparent [46]. In particular, the chapter starts with a description of the taxonomy of XAI, in order to show the different possibilities. Then the chapter shows in more details the frameworks used in this thesis such as `xclingo`, xASP, SHAP and LIME.

4.1 Introduction

AI systems are typically considered black boxes, meaning their internal workings are not easily interpretable. Indeed, the primary challenge that XAI seeks to address is this opacity. However, the aim is not solely to make these systems more transparent, but also to tackle related challenges, such as developing techniques that enable effective human interaction and ensuring the trustworthiness of the systems' inferences. This is particularly valuable for domain experts, such as medical professionals, who not only require assistance in solving specific tasks but also need to understand and trust the system's output. It is equally important for developers: when a system produces an incorrect result, having an explanation can help trace the origin of the error and guide subsequent debugging or model improvement. Moreover, XAI helps in complying with legal obligations. In fact, new regulations, like the GDPR [22], require that people are

given clear information about how decisions made by AI systems work. Furthermore, the recently adopted EU AI Act (2024) introduces a risk-based regulatory framework for artificial intelligence, effectively functioning as a product safety law that enforces the design of trustworthy and transparent AI systems. Notably, the AI Act stipulates that AI systems must be developed and deployed in a manner that enables appropriate traceability and explainability, thereby complementing the rights and obligations established by the GDPR.

One of the fundamental challenges XAI faces is the black-box nature of many modern machine learning models, particularly deep neural networks. These models have achieved remarkable predictive accuracy but are extremely complex, often involving millions or even billions of parameters. The internal representations and the flow of data through their layers are difficult to inspect. In particular, AI systems can be categorised into three classes based on their interpretability and transparency:

- *Black*-box models are those whose internal logic and decision-making processes are not accessible or understandable to humans. These models, which often include deep neural networks, typically achieve high predictive accuracy but offer little to no insight into how their outputs are generated.

- *Gray*-box models represent a middle ground: while their internal mechanisms remain partially opaque, it is possible to extract and interpret some aspects of their reasoning with reasonable accuracy.

- *White*-box models are fully transparent systems whose decision-making logic is explicit and easily understood. However, these models generally exhibit lower predictive accuracy compared to their black-box counterparts.

Another major challenge lies in the interpretability-accuracy trade-off. It is widely recognised that more interpretable models often sacrifice predictive power, whereas highly accurate black-box models offer little insight into their reasoning. Another problem that is still open is how to evaluate explainability. There is no universally accepted metric for assessing explanation quality, and current evaluation methods often rely on subjective human judgment. Metrics such as *fidelity* (how well the explanation reflects the model) or *plausibility* (degree to which something is believable) do not necessarily guarantee that an explanation is understandable or trustworthy to humans.

Scalability is another pressing issue. Generating explanations for large-scale models and high-dimensional datasets (such as images or text) is computationally expensive. Human and ethical considerations also play a critical role in XAI. Since explanations are meant for human users, they must take into account how individuals perceive, trust, and respond to them. Flawed explanations may unjustifiably increase trust in imperfect systems. Moreover, ethical risks must be considered, including the possibility of unintentionally revealing sensitive information or reinforcing social biases.

4.2 Taxonomy

The evolution of AI has been marked by alternating periods of rapid progress and stagnation. These shifts have been driven by advancements in computational power, the

development of novel software techniques, and the persistent challenge arising from the lack of necessary computational power and interpretability of the results, trying also to make AI systems more interpretable and understandable to human users [47]. Indeed, as AI models became more sophisticated, their decision-making processes became more opaque, creating difficulties in understanding and justifying their outputs. Early AI models, such as decision trees and rule-based systems, were relatively easy to interpret, as their logic was transparent and human-readable. However, modern AI systems, particularly deep learning models, are real black-box models.

To address the interpretability challenge, researchers have explored many strategies [48, 49]. XAI techniques can be classified along several dimensions:

1. **Intrinsic vs post-hoc explainability**.
   - *Intrinsic* methods involve models that are inherently interpretable, such as decision trees or rule-based systems.
   - *Post-hoc* methods apply interpretability techniques to black-box models after training in order to make their decisions more understandable.
2. **Reverse engineering approaches**. This perspective focuses on analysing the outputs of opaque systems and reconstructing explanations to clarify their underlying logic. Three main goals can be identified [50]:
   - Explaining the BB model by developing surrogate models that emulate its behaviour in a human-understandable way.
   - Explaining the BB outcomes by generating explanations for individual predictions without necessarily revealing the BB logic for all possible inputs. This approach provides insight into why a particular decision was made, rather than how the entire model operates.
   - Inspecting the BB model by creating visual or textual representations of how inputs are processed and why certain predictions are more likely than others. Such representations assist in diagnosing errors, detecting biases, and improving model reliability.
3. **Global vs local explanations**. Explanations may differ in scope:
   - *Global* explanations aim to provide insights into the overall model behaviour.
   - *Local* explanations instead focus on clarifying individual predictions.
4. **Types of explanation provided**. Methods can also be categorised according to the explanatory form they produce, including:
   - Feature attribution methods (e.g., SHAP, LIME).
   - Example-based explanations (e.g., counterfactuals, prototypes).
   - Symbolic, rule-based techniques that translate model decisions into logical expressions.
5. **Model-agnostic vs model-specific methods**. The distinction depends on the degree of dependence of the explanation algorithm on the underlying model.

- *Model-agnostic* methods make no assumptions about the architecture or functioning of the model and rely only on its input–output behaviour.
- *Model-specific* methods exploit the internal structure, parameters, or properties of the model, thereby providing more faithful and detailed explanations of its functioning.

6. **Prototypical and concept-based methods**. These methods provide explanations by appealing to human-friendly concepts or abstractions rather than low-level input features. In doing so, they resemble human reasoning more closely than standard XAI approaches.

7. **Purpose of the explanation**. XAI techniques can also be classified according to their primary objective:

   - *Debugging*-oriented techniques aim to identify sources of inconsistencies or errors in the models, enabling users to fix them. For instance, in the field of ASP, such techniques help detect erroneous rules, incorrect assumptions, or missing constraints that may cause unintended behaviour.
   - *Justification*-oriented techniques aim to explain why a certain outcome was made. For example, they can be used to explain why literals are present or absent in an answer set. These explanations are particularly valuable for non-expert users, who may not be familiar with the inner workings of logic programs but still require an understanding of the system's outputs.

4.3 Symbolic Knowledge Extraction

Symbolic Knowledge Extraction (SKE) techniques, which will be used as shown in Sections 6.2, 6.3 and 6.4, have become increasingly important in AI applications as they offer a means of explaining BB predictors. The term "*symbolic*" refers to the way knowledge is represented: specifically, it denotes languages that are both intelligible to humans and interpretable by machines. This comprehends a variety of logical formalisms, while excluding the fixed-size numerical tensors typically used in sub-symbolic machine learning.

SKE refers to the process of transforming the knowledge acquired by sub-symbolic predictors, such as neural networks, into a symbolic representation, for instance in the form of rules or logical statements. The main purpose of this transformation is to provide explanations for models that are otherwise difficult to interpret, thereby enabling human designers to inspect and understand their internal reasoning and behaviour. Despite the availability of numerous methods for performing SKE, the fundamental notions of it are seldom discussed in general terms within the scientific literature, which often focuses on specific techniques or case studies rather than on a unified conceptual framework.

Symbolic knowledge extraction plays a dual role. First, it enhances transparency, as the extracted rules explicitly show which conditions lead to a certain prediction. Second, it supports trust and validation, since domain experts can inspect, validate, and even contest the rules, which is not possible with purely numerical attribution methods. Nevertheless, symbolic knowledge extraction also presents challenges. Rule sets may become complex or too wide when dealing with high-dimensional real-world data.

4.4 Textual and Visual Explanations

In this thesis, since we are focused on ASP, we employ `xclingo` and XASP2 [51]. The first one in applied only to the juridical domain (Section 6.3), while the other is applied on weather and veterinary domains (Sections 6.2 and 6.4)The choice of these tools is motivated by their ability to generate structured, comprehensible justifications for the conclusions derived from ASP models. However, to fully understand how these techniques operate and to appreciate the principles behind their explanatory capabilities, it is essential to introduce two foundational approaches that support our methodology: *off-line* justification and *causal graph* justification. SKE is an important strategy within black-box reverse engineering, specifically falling under the category of model explanation. The process involves taking a trained and optimised black-box model, along with the original training data, and applying a knowledge extraction procedure. The output of this process is a human-interpretable predictor that approximates the decision-making behaviour of the BB model. SKE techniques can be categorised into three primary approaches [52]: *decompositional*, *pedagogical*, and *eclectic*. Each of these methods differs in how they interact with the underlying BB model and the extent to which they rely on its internal structure. While decompositional techniques exploit the model's inner workings, pedagogical approaches treat the BB as an opaque function to be approximated, and eclectic strategies combine elements of both.

Decompositional methods extract knowledge from opaque models by directly analysing their internal logic, structure, and parameters. These techniques are tightly coupled with the specific class of the underlying predictive model, requiring dedicated extraction algorithms tailored to the BB's nature. The main advantage of decompositional strategies is that they can exploit detailed information about the BB's decision-making process, enabling more precise and faithful explanations. The main drawback of decompositional methods is their dependency on the architecture of the BB model. Each machine learning paradigm requires a specific extraction technique, reducing the generality of the approach. Additionally, if the BB undergoes structural modifications, the corresponding extraction process must be adapted accordingly.

Pedagogical SKE techniques, which have been used in our study, make no assumptions about the BB's architecture or internal parameters. Since those techniques do not rely on knowledge of the BB's internal workings, they can be applied to any predictive model, making them highly flexible and widely applicable. These techniques generally involve querying the BB with a diverse set of input instances and capturing the corresponding output responses. A new, interpretable model, that in our case is a logic program, is then trained to replicate the BB's input-output behaviour as closely as possible. The resulting surrogate model serves as an explanation of the BB's decision process.

Eclectic SKE techniques integrate aspects of both decompositional and pedagogical methods, aiming at balancing accuracy and generality in their explanations. While they require some level of access to the BB's internal structure, they also incorporate strategies that generalise across different architectures.

4.4.1 Off-line Justification

Pontelli and Son [53] introduce the concept of off-line justification, a method for explaining the truth value of an atom within a specific answer set A of a ground ASP program P. This justification is represented as a labelled, directed graph, where nodes correspond to annotated ground atoms and edges illustrate logical dependencies.

Each node in the justification graph is annotated as either a^+ (indicating that $a \in A$) or a^- (denoting that $a \notin A$), alongside special labels such as *assume*, $\top$, and $\bot$. Edges capture how the truth value of an atom is supported by other atoms within the answer set through the rules of P. Labels on edges, either $+$ or $-$, reflect whether the dependency arises from a positive or negative occurrence of an atom in a rule. Intuitively, *assume* is used for atoms we are not seeking any explanations for, $\top$ justifies the truth of facts, and $\bot$ explains the falsity of atoms that do not appear as rule heads in the program.

Formally, an *explanation graph* is described as follows.

Definition 4.4.1 (Explanation Graph)**.** *For a program P, an* explanation graph *(or* e-graph*) is a labelled, directed graph (N, E), where $N \subseteq A^+ \cup A^- \cup \{assume, \top, \bot\}$ and $E \subseteq N \times N \times \{+, -\}$, which satisfies the following properties:*

- *the only sinks*[1] *in the graph are: assume, $\top$, and $\bot$;*
- *for every $b \in N \cap A^+$, we have that (b, assume, -) $\notin E$ and (b, $\bot$, -) $\notin E$;*
- *for every $b \in N \cap A^-$, we have that (b, assume, +) $\notin E$ and (b, $\top$, +) $\notin E$;*
- *for every $b \in N$, if (b, l, s) $\in E$ for some $l \in$ assume, $\top$, $\bot$ and $s \in +$, - then (b, l, s) is the only outgoing edge originating from b.*

Example 4.4.1
We give a brief example on how to read e-graphs. Consider the one in Fig. 4.1: it represents the truth value of p by establishing its positive dependency on the truth values of q and r. In this structure, q is assumed to be true, while r is a fact explicitly stated in the program.

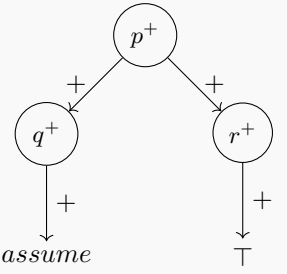


Figure 4.1: Examples of e-graphs

Consider now the following program P:

[1] In a directed graph, a sink node is a vertex with no outgoing edges.

```
p :- q.
q :- r, s.
r :- not t.
s :- not t.
```

The justification of p^+ can be represented with the e-graph below (Fig. 4.2). Here we have that p^+ is positively supported by truth value of q, which itself is positively supported by the truth values of r and s. Both r and s derive their support from falsity of t. Since t does not appear as the head of any rule in the program, its falsity is justified by $\bot$, representing the absence of a derivation.

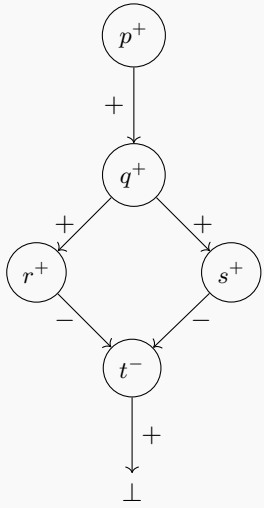


Figure 4.2: Examples of e-graphs

Once the e-graph is computed, it is possible to extract from it the elements that are implied in proving the truth value of the atom. This set is called *support* of the atom.

Definition 4.4.2 (Support)**.** *Let $G = (N, E)$ be an e-graph and $b \in N \cap (A_p \cup A_n)$ a node in G. The direct* support *of b in G, denoted by $support(b, G)$, is defined as follows:*

- $support(b, G) = \{atom(c) \mid (b, c, +) \in E\} \cup \{not\ atom(c) \mid (b, c, -) \in E\}$*, if for every $l \in \{assume, \top, \bot\}$ and $s \in \{+, -\}, (b, l, s) \notin E$;*
- $support(b, G) = \{l\}$ *if $(b, l, s) \in E, l \in \{assume, \top, \bot\}$ and $s \in \{+, -\}$.*

Example 4.4.2
In the Example 4.1, for the graph on the right, let's call it G, we can compute the following support sets:

- $support(p^+, G) = \{q\} \cup \{\neg s\} = \{q, \neg s\}$
- $support(s^-, G) = \{\bot\}$

A key limitation of off-line justification graphs is that they do not explicitly reference the rules applied in the reasoning process. While this abstraction can simplify the representation by avoiding redundant explanations when multiple rules lead to the same conclusion, it may also lead to intelligibility. Users seeking to fully comprehend the justification must manually reconstruct the logical derivation by tracing the dependencies back to the program's rules.

4.4.2 Causal Graph Justification

Causal graphs, introduced by Cabalar et al. [28], provide a structured representation of causal relationships between variables, enabling formal reasoning about cause and effect. In contrast with off-line justifications, causal graph justifications do not show any negative literals since they are assumed to hold by default.

The semantics for causal justifications is based on a multi-valued extension of the answer set semantics. In this framework, each true literal in a model is associated with a set of causal values that express the causal reasons for its inclusion. These causal values, which represent sets of causal justifications, can be depicted as causal graphs, providing a clear visual representation of the causal dependencies and relationships that lead to the inclusion of specific literals in the answer set.

Causal justifications are defined in terms of the causal value assigned to each atom in a causal answer set. Unlike traditional answer sets, which assign truth values (true or false) to atoms, causal answer sets assign causal values to each atom, capturing the underlying reasons behind their inclusion. These causal values form a completely distributive (and complete) lattice, which serves as the foundation for the multi-valued extension of the answer set semantics.

These graphs are typically modelled as DAGs, where nodes represent variables, and directed edges indicate causal dependencies.

Definition 4.4.3 (Causal Graph)**.** *Given some set L of (rule) labels, a causal graph (*c-graph*) G is a reflexively and transitively closed directed graph, $G^* = G$, whose vertices are labels,* i.e. *$V \subseteq L$. We denote by C_L the set of all possible causal graphs over L.*

Intuitively, the vertices in the justification graph represent the rules involved in deriving a given atom (or formula), while the edges establish a partial ordering that reflects the sequence in which these rules were applied.

Given that we can define an ordering relation among graphs as follows.

Definition 4.4.4 (Sufficient)**.** *A causal graph G is* sufficient *for another causal graph G', written $G \leq G'$, when $G \supseteq G'$*

Saying that G is sufficient for G' intuitively means that G contains enough information to achieve the same effect as G', but potentially with more information than necessary. In this sense, G' might be considered more detailed, but G still captures all the essential properties necessary to replicate the outcomes of G'.

4.4.3 xclingo

Cabalar et al. [54] developed a tool called `xclingo`, which generates explanations from ASP programs. The core idea is to apply causal graph justification to produce a tree-like

structure that traces the causes leading to the computed answer set. This structure is then rendered in natural language, providing a human-readable explanation.

The system extends the syntax of `clingo` by allowing special annotations that enrich ASP programs with additional explanatory information. These annotations take the form of comments that are therefore ignored by `clingo`, thus ensuring compatibility with standard ASP syntax. Two main annotation types are supported. The first one, `%!trace`, is used to label specific atoms for inclusion in the explanation, while the second one, `%!trace_rule`, is applied to rules. Both annotations can include a format string to customise how their content is described in natural language. While `trace_rule` annotations associate labels with particular derivations, using them exclusively can make the program verbose and harder to maintain. In such cases, `trace` annotations are preferable, as they permanently associate a label with an atom, independent of the rules that produced it. Finally, there is a third kind of annotation that is used to specify which atoms should be included in the output explanations. The annotation used in this case is `%!show_trace` which function is similarly to the `#show/1` directive in `clingo`, but targets the explanation graph.

Example 4.4.3
Consider the following code with `xclingo` annotations:

```
%!trace "% is sunbathing", X sunbathing(X).
sunbathing(X) :- weather(W), outside(X), W="sunny".
%!trace_rule "% is protected from UV rays", X
protected(X) :- use(X, C), creamSpf(C).
%!trace_rule "% got sunburnt because was not protected", X
sunburnt(X) :- not protected(X), sunbathing(X).
%!trace_rule "% got tanned", X
tanned(X) :- protected(X), sunbathing(X).

%!trace "% is outside", X outside(X).
%!trace "The weather is %", X weather(X).
%!trace "% used %", X, C use(X, C).
%!trace "% is a cream with spf", C creamSpf(C).

%!show_trace sunburnt(X).
%!show_trace tanned(X).
```

If we state as facts that:

```
outside("Alice").
weather("sunny").
```

and we run the code we will get:

```
Answer 1
  *
  |__Alice got sunburnt because was not protected
  |  |__Alice is sunbathing
  |  |  |__Alice is outside
  |  |  |__The weather is sunny
```

Note that in this case, we had to include the reason for sunburn in the rule's annotation, since the negation of atoms cannot be directly tracked. Indeed, only predicates can be annotated, as shown in lines 10–13 of the code presented earlier. For this reason, if no information is available regarding Alice's use of sunscreen (and we therefore assume she does not use it), we cannot generate a statement such as "*Alice is not using sunscreen*".

Instead, if we also provide:

```
use("Alice", "sunscreen").
creamSpf("sunscreen").
```

Then we will get:

```
Answer 1
  *
  |__Alice got tanned
  |  |__Alice is sunbathing
  |  |  |__Alice is outside
  |  |  |__The weather is sunny
  |  |__Alice is protected from UV rays
  |  |  |__Alice used sunscreen
  |  |  |__sunscreen is a cream with spf
```

The computation in `xclingo` begins with the translation of the annotated ASP program P into an equivalent logic program P', which is equivalent to P without annotations, but which incorporates auxiliary predicates and theory atoms. These additions enable tracking of rule applications during reasoning. This translation proceeds in two phases:

1. the `trace` and `trace_rule` annotations are converted into theory atoms, which are passed through `clingo`'s grounder and later processed via the Python API,

2. `show_trace` annotations are transformed into traditional rules that introduce auxiliary predicates (e.g., `show_all_p`) corresponding to each predicate p marked for explanation.

During this process, additional information about each annotated rule is recorded, including: the original rule head, the rule body, and the variable names used in the triggered rules.

Once the program P' is generated, `xclingo` invokes `clingo`'s solver to obtain the answer sets. For each answer set, it constructs an explanation by analysing the causal data collected during solving. This involves building a *causes table*, which contains one row per rule and records the trace labels and rule bodies responsible for each derived atom. The final step is to filter this table according to the `show_trace` specifications, identifying the atoms for which explanations should be produced.

The result is a causal justification graph, in which nodes and edges are annotated with their natural language labels, clearly showing the causal dependencies among atoms. This graph is rendered directly in the terminal, allowing users to visually follow the derivation of each explained atom.

4.4.4 xASP2

In the field of logic programming and explainability, xASP2 [27] is designed to enhance the interpretability of ASP by offering a structured approach to generating explanations for ASP-derived solutions. Specifically, xASP2 establishes a connection between the presence or absence of an atom in an answer set and the logical rules responsible for its derivation. While alternative systems, such as `xclingo` [54], DiscASP [55], xASP [56], and exp(ASPc) [57], could also be employed for this purpose, xASP2 was chosen due to its ability to address critical aspects that other systems lack, including, unlike `xclingo`, the explanation of false atoms (see Example 4.4.4) and support for specific advanced language constructs. Although these capabilities may not be strictly necessary for the current application, xASP2 provides a more adaptable foundation that helps future extensions and improvements.

In particular, for each atom in the answer set, xASP constructs a DAG in which the node corresponding to the atom is annotated as "*explained by REASON*". The label *REASON* denotes the underlying justification for the truth value of the atom and can take one of the following forms:

- `assumption`: atoms that are assumed to be false are explained by assumption;
- `initial_well_founded`: atoms that are part of the initial well-founded model are explained by this status;
- `(support, rule)`: true atoms are explained by a supporting rule whose body literals have already been explained are explained by support;
- `lack_of_support`: false atoms are explained by the fact that all potentially supporting rules have already been explained;
- `(required_to_falsify_body, rule)`: a false atom is explained by a rule with a false head that includes the atom in its body, provided that all other body literals are true;
- `(choice_rule, rule)`: a false atom is explained by a choice rule with a true body, where the number of true head atoms already reaches the upper bound of the choice.

Example 4.4.4
Consider the following ASP program to solve the n-queens puzzle:

```
coord(1..4).
4{queen(X, Y): coord(X), coord(Y)}4.

wrong :- queen(X1,Y), queen(X2,Y), X1 != X2.
wrong :- queen(X,Y1), queen(X,Y2), Y1 != Y2.
wrong :- queen(X1,Y1), queen(X2,Y2), X1!=X2,|X1-X2| =|Y1-Y2|.

:- wrong.
```

Running xASP on this input would produce very large graphs; to keep the example clear, we added a few additional facts that reduce the complexity of the resulting graph.

```
queen(3,4). queen(1,3). queen(2,1).
```

xASP produces multiple graphs for this program, as the position of the last queen can be justified in different ways. Below, we report some of these variations.

In the first graph (Fig. 4.3) we can see that the presence of a queen in position $(4, 2)$ excludes the placement of another queen in $(2, 4)$ and $(4, 4)$. Allowing both positions would entail the derivation of the atom `wrong`, which represents a constraint violation.

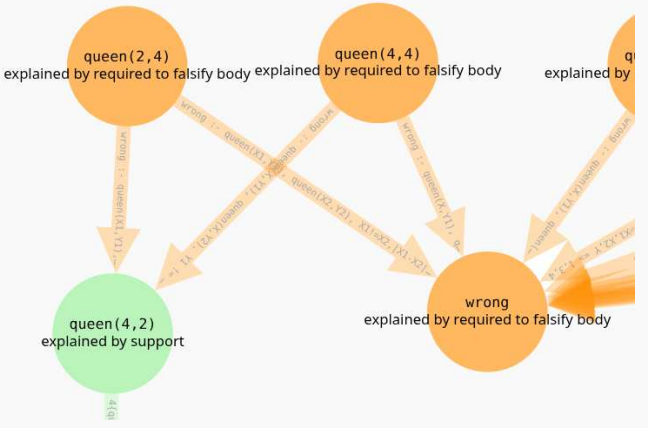


Figure 4.3: Portion of DAG that explains why `queen(2,4)` and `queen(4,4)` is false.

In the second one (Fig. 4.4), instead, we can see that the position $(3, 1)$ is not permitted. In this explanation, it is a consequence of the choice rule, which ensures that exactly four queens must be placed on the board.

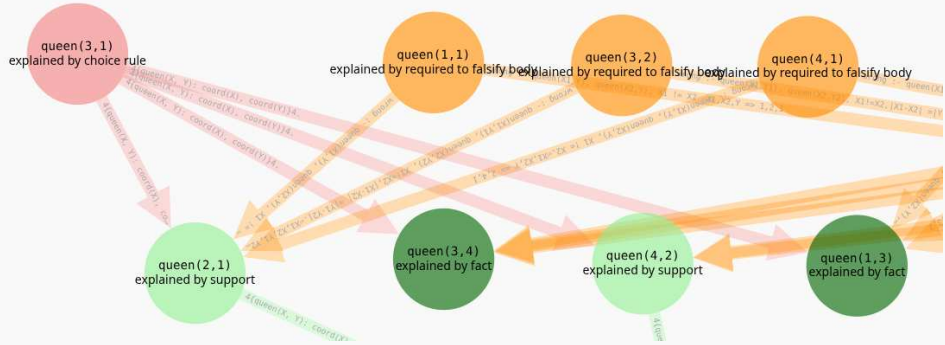


Figure 4.4: Portion of DAG that explains why `queen(3,1)` is false.

For the last case (Fig. 4.5), we report the complete graph. As it is rather large, it cannot be easily read in detail, so we provide a description instead. The node at the top (the bright red one connected to all the other light red nodes) is labelled `wrong` and is explained by `lack_of_support`, meaning that there is no rule supporting the atom `wrong`. The remaining nodes (the light red ones) correspond to all the queen positions that are not allowed. Their falsity is justified by `choice_rule`, indicating that once the upper limit of four true queens is reached, all the remaining candidate positions are automatically set to false.

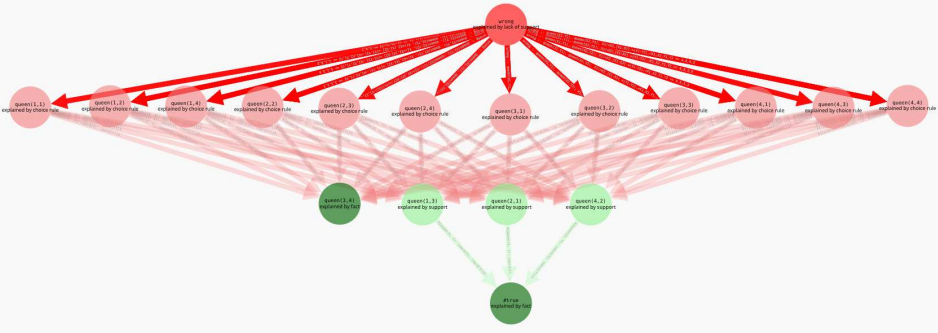


Figure 4.5: DAG with nodes of type `lack_of_support` and `choice_rule`

Given a program Π, an answer set A, and an atom α, xASP2 is designed to answer queries such as "Why does $\alpha \in A$ (or $\alpha \notin A$)?" by identifying the relevant subset of Π that justifies the atom's inclusion or exclusion. Explanations are represented as justification graphs, specifically DAG, where nodes correspond to atoms and edges denote logical rules. An edge from an atom to a rule signifies that the atom appears in the rule's body, while an edge from a rule to an atom indicates that the rule derives that atom. For each atom in a stable model, xASP2 determines a supporting rule whose body is satisfied and composed only of previously explained atoms, thereby maintaining the graph's acyclicity.

Two primary challenges arise in constructing such explanations: (i) identifying a

minimal set of assumptions that sufficiently justify the inclusion of α in A, and (ii) managing complex ASP constructs, such as choice rules and aggregates, to facilitate intuitive explanations for why certain atoms are absent. While answer set semantics permit explanations based on the assumption of false atoms, this often results in explanations that are overly general, reducing them to mere assumptions for all false atoms.

To initiate the explanation process, xASP2 first performs a preprocessing step to derive justifications for a grounded program P. This computation begins with a three-valued interpretation, represented as (I^+, I^-), where both sets are initially empty. A three-valued interpretation consists of a pair (L, U), where L and U are sets of ground atoms such that $L \subseteq U$ and neither contains the atom $\bot$. The objective is to iteratively expand these sets to determine the truth values of atoms based on the program's rules.

The set I^+ is expanded through iterative inference: a rule's head is inferred only if every atom in its body is determined, *i.e.*, $b \in I^+$ or $b \in I^-$. Meanwhile, the set I^- is constructed by adding atoms belonging to an unfounded set $X \subseteq B_P$. An unfounded set X is defined for a ground logic program P such that, for every atom $a \in X$ and each rule $r \in P$ with head a, at least one of the following conditions holds: (i) either some literal in the rule's body is known to be false, that is, if b appears positively in the body, then $b \in I^-$, and if b appears negatively, then $b \in I^+$, or (ii) an atom b appearing positively in r satisfies $b \in X$. The purpose of expanding I^- is to remove self-referential sets of atoms, where each atom is justified by another within the set, yet none can be independently derived as true. These cycles would otherwise undermine meaningful explanations, so xASP2 ensures only valid, acyclic justifications remain.

Once preprocessing is complete, xASP2 constructs an explanation graph that resembles the *offline justification* approach, which visualises the relationships between atoms and inference rules within the program. Throughout the derivation process, xASP2 tracks the rules applied at each step and annotates the corresponding edges in the explanation graph accordingly.

Example 4.4.5
Consider again the program of the Example 4.4.3. If we use xASP2 to get an explanation of the first case (that was the one in which Alice got sunburnt) we get a graph in Fig. 4.6.

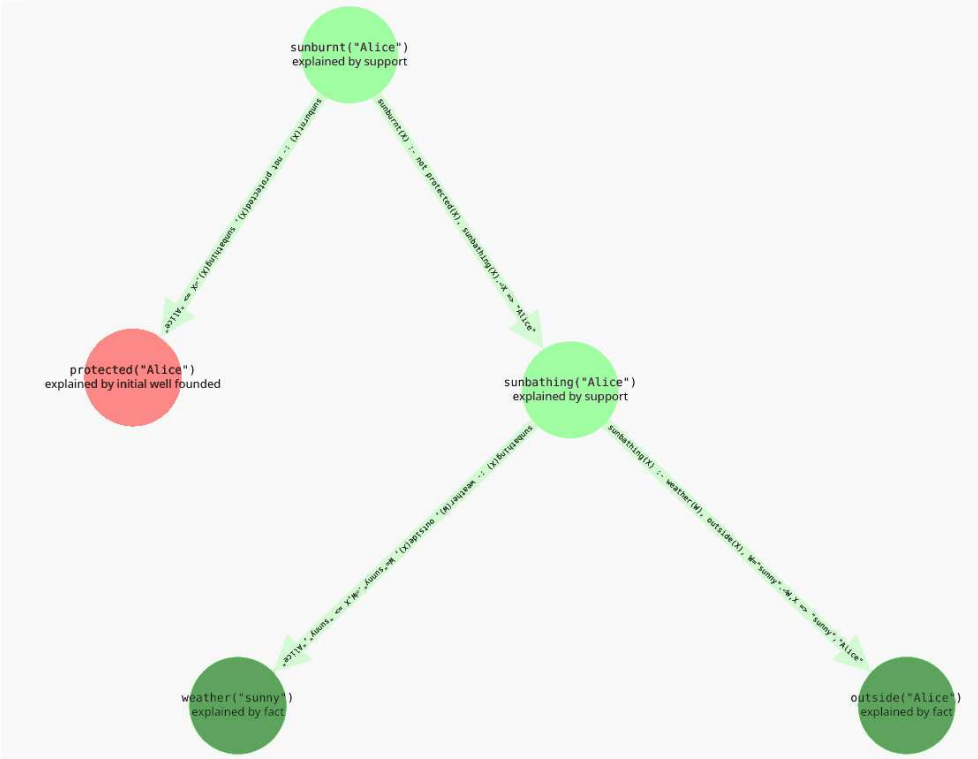


Figure 4.6: DAG generate from xASP to explain `sunburnt("Alice")`

Finally, xASP2 employs a metaprogramming approach via the Clingo Python API, offering a flexible Python interface. Users can generate explanations for individual literals or request graphs for entire answer sets. Additionally, xASP2 features an interactive, web-based visualisation tool that allows users to explore and analyse the generated explanation graphs dynamically.

4.4.5 Model-Agnostic Explanation Approaches

Post-hoc model-agnostic explanation methods have gained relevant attention as tools for improving the interpretability of complex machine learning models. Among these, LIME [58] and SHAP [59] have emerged as one of the most widely adopted techniques and we decided to test them in the veterinary medicine field. A complete survey by Saw et al. [60] highlights LIME and SHAP as representative approaches frequently employed in practice. More recently, a critical comparison by Salih et al. [61] demonstrated that the outcomes of both methods are highly dependent on the underlying machine learning model and are particularly sensitive to feature collinearity. LIME has proven especially useful in medical imaging, where its explanations highlight regions of an image with high importance scores, pointing to the areas that most strongly influence a model's prediction. This property has enabled applications such as explaining COVID-19 and pneumonia predictions from X-ray data [62], as well as interpreting Parkinson's disease

classification from SPECT DaTSCAN datasets [63]. SHAP, in contrast, offers both local and global interpretability: it can attribute the role of individual features in a single prediction while also capturing their aggregated contribution across an entire dataset—something not achievable with LIME. In the medical field, SHAP has been applied in several successful studies, including the work of Rahman et al. [64], who integrated SHAP into an AI system for MRI-based brain tumour detection, and the study of Nahiduzzaman et al. [65], who employed SHAP for lung cancer classification. In our work LIME and SHAP will be used as comparison with our results.

LIME

LIME's core idea is to construct an interpretable surrogate model that approximates the behaviour of an underlying black-box model. To do so, LIME generates a large number of perturbed versions of the instance to be explained. In the case of image data, for example, this may involve masking or blurring parts of the image. Each perturbed instance is then passed through the black-box model, and the corresponding predictions are recorded. By analysing how the predictions change in response to these perturbations, LIME identifies which parts of the input are most influential.

Since not all perturbed samples contribute equally to the explanation, LIME assigns greater weight to those that are more similar to the original instance. It then trains a simple, interpretable model, such as a linear regression, on these weighted samples. Note that this surrogate model is fitted only in the neighbourhood of the instance of interest, as the goal is to capture the local behaviour of the black-box model rather than its global decision boundary. The result is a model that is easy to interpret and that highlights the features most relevant in the local region around the chosen instance, though not necessarily for the precise instance itself.

Formally, the explanation for a data point x is the model g that minimizes the locality-aware loss L measuring how unfaithful g approximates the model to be explained $\hat{f}$ in its vicinity π_x while keeping the model complexity $\Omega(g)$ low:

$$explanation(x) = arg \min_{g \in G} L(\hat{f}, g, \pi_x) + \Omega(g)$$

SHAP

The primary objective of SHAP is to explain the prediction of an individual instance by attributing to each feature its contribution to the final outcome. SHAP is grounded in the concept of Shapley values, a classical notion from cooperative game theory. In this framework, the prediction task is viewed as a game in which features act as players, each contributing to the result. Shapley values provide a principled way of distributing the overall outcome fairly among the features, thereby quantifying their relative importance.

Computing exact Shapley values, however, is computationally prohibitive. SHAP therefore relies on efficient approximations, which, despite being faster, can still be computationally demanding for large or complex models. Nevertheless, these approximations allow SHAP to capture both the direction and magnitude of each feature's influence on a specific prediction. A positive Shapley value indicates that a feature pushes the prediction towards the examined label, while a negative value indicates that it pushes the prediction away from it.

Formally, an explanation for a prediction $f(x)$ is:

$$g(z') = \phi_0 + \Sigma_{i=1}^{M} \phi_i z'_i$$

where g is the explanation model, z' is a binary vector representing feature presence/absence, M is the number of input features, ϕ_0 is the base value (*i.e.* average model output over the training data), and ϕ_i is the Shapley value for feature i.

4.4.6 Evaluating Explainability

The evaluation of extracted knowledge from machine learning models involves several key properties, though their quantitative assessment is often challenging due to subjectivity and the lack of universally accepted definitions [66, 67]. Indeed, there are no widely acknowledged or formally established strategies for achieving a unified, quantitative assessment of these properties.

The key properties to consider are:

- *Predictive Performance*: it measures how well predictions align with the ground truth. When assessed against the original BB model's predictions, it is referred to as *fidelity*. Both predictive performance and fidelity depend on the task and are typically evaluated using standard metrics such as accuracy, F1 score or mean absolute error (for the definitions of those metrics see Section 5.3.3).

- *Readability*: it is often estimated based on the size of the extracted knowledge. For rule-based models, this may be the number of rules, while for decision trees, it is the number of leaves. Further improvements consider factors like the number of antecedents per rule or internal nodes.

- *Completeness*: it measures whether the extracted knowledge can provide an output when queried, regardless of its prediction quality. It estimates the likelihood that the knowledge will produce a response.

- *Robustness and Consistency*: they often overlap [68]. Robustness refers to a technique's ability to generate reliable predictions across different training sessions, while consistency measures how resilient the extracted knowledge remains when the training data undergoes minor perturbations.

- *Fairness*: it evaluates whether the extraction procedure ensures unbiased outputs, avoiding favoritism toward specific classes. Defining and quantifying fairness remains an open challenge [69, 70]. However, since extraction techniques expose the internal reasoning of black-box models, they reflect the fairness—or bias—present in the original model's predictions, making them useful for identifying biases in opaque predictors.

Despite the lack of a unified, quantitative assessment method for knowledge quality, in the thesis of Sabbatini [71] three formal scoring metrics have been introduced: Q_s, *FIdelity vs. REadability* (FiRE) and *Index for Complete quality Evaluation* (ICE).

The Q_s index [72] is a naive scoring function that evaluates symbolic knowledge quality based on three loss components:

- *predictive* loss (p_{loss}), which is equal to $1 - R^2$ for regression tasks, where R^2 is the coefficient of determination, or $1 - accuracy$ for classification tasks,
- *readability* loss (r_{loss}), which take into account the human-readability degree of the knowledge and is equal to the rule amount r,
- *coverage* loss (c_{loss}), which measure the completeness of the extracted knowledge; it is computed as $2 - cov$, where cov represents dataset coverage in $[0, 1]$.

The Q_s score is defined as:

$$Q_s = p_{loss} \times r_{loss} \times c_{loss} = (1 - R^2) \times r \times (2 - cov). \tag{4.1}$$

High-quality knowledge bases exhibit small values for p_{loss}, r_{loss}, and c_{loss}, leading to a low Q_s score. However, Q_s does not incorporate user feedback, making it not flexible.

To address the limitations of Q_s, a more sophisticated metric, FiRe [73], was introduced. The FiRe score is a multivariate function defined as:

$$FiRe(\psi, p_{loss}, r_{loss}) = p_{loss} \cdot \lceil \frac{r_{loss}}{\psi} \rceil \cdot r_{loss}^{0.05} \tag{4.2}$$

where $\lceil \cdot \rceil$ is the ceiling function, and ψ is a user-defined parameter that controls the trade-off between fidelity and readability. Setting $\psi = 1$ assigns equal importance to predictive and readability losses, while increasing ψ reduces the impact of readability on the overall quality assessment.

The ICE [74] score provides a more complete assessment by incorporating predictive performance, readability, and completeness, while allowing user-defined weighting parameters. ICE is formulated as:

$$ICE(p, r, c, \phi, \rho) = P(p, \phi) \cdot R(r, \rho) \cdot c \tag{4.3}$$

where p, r, and c correspond to predictive performance, readability, and completeness, respectively, and ϕ and ρ determine the relative importance of predictive performance and readability. The functions $P(\cdot)$ and $R(\cdot)$ are generalised sigmoid functions ensuring a continuous and differentiable quality evaluation. By integrating user-defined parameters, the ICE score offers a flexible and robust method for assessing knowledge quality across different applications.

5

Artificial Neural Networks

This chapter provides a brief introduction to Artificial Neural Networks, a foundational concept in deep learning inspired by the biological nervous systems. ANNs mimic how the brain processes information using layers of interconnected artificial neurons to solve complex problems like image recognition, language understanding, and predictive modelling. Given the broad scope of this subject, we will focus specifically on the aspects of ANNs that are relevant to this work, offering a concise explanation of the concepts and techniques employed.

5.1 Introduction

The story of ANNs begins in the 1940s, rooted in the intersection of neuroscience, mathematics, and computer science. Inspired by how the human brain processes information, Warren McCulloch and Walter Pitts published a seminal paper in 1943 [75] that proposed a mathematical model of a neuron. Their model demonstrated that networks of such artificial neurons could, in theory, perform logical operations, contributing to the birth of computational neuroscience and early artificial intelligence.

Over the decades, ANNs evolved from these theoretical foundations into powerful tools for solving real-world problems. Milestones like Frank Rosenblatt's Perceptron in the 1950s [76] and Seppo Linnainmaa's development of backpropagation in 1970 laid the groundwork for modern neural network training methods. These advances, combined with the rapid growth of computational power and access to large datasets, have propelled ANNs to the forefront of AI innovation.

Today, ANNs are at the heart of state-of-the-art AI research and applications, with architectures like Transformers redefining fields such as natural language processing and computer vision. The journey of neural networks reflects not only the evolution of technology but also a growing understanding of how learning systems can emulate aspects of human cognition.

At their core, artificial neural networks are computational models designed to mimic the way the human brain processes information. They consist of layers of interconnected

nodes, called *neurons*, that work together to identify patterns and relationships in data. Each connection between neurons has a weight, which determines how much influence one neuron has on another. By adjusting these weights during training, the network learns to map inputs to outputs, enabling it to perform tasks like classification, prediction, and decision-making. This ability to learn from examples and generalize to unseen data is what makes ANNs so powerful and versatile.

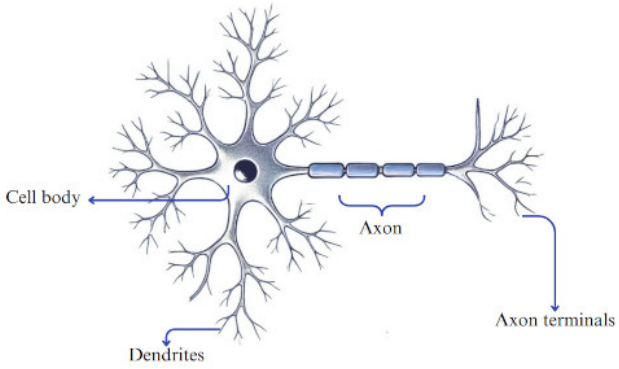
Despite their success, ANNs come with their set of challenges. Training deep networks requires substantial computational resources, and ensuring that models generalize well to unseen data is an open issue at the moment. Moreover, the *black-box* nature of ANNs can make their predictions difficult to interpret, posing challenges in domains where explainability is crucial.

5.2 Basic Components

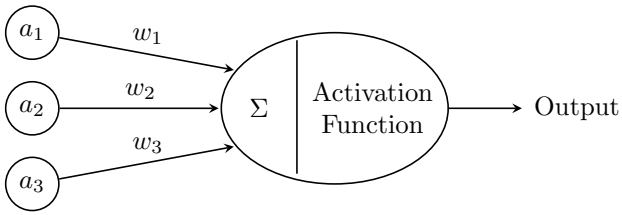
As just said, an Artificial Neural Network is a computational model inspired by biological neural networks. It is composed by interconnected neurons, functioning as processing units, and connections with adjustable weights. Through training algorithms, these weights are tuned based on input data, culminating in a discriminating function which is able to classify different inputs. The training process is typically iterative, and it involves feeding the network examples from the training dataset and adjusting the weights based on the network's output. Indeed, in general, the implementation of an AI model begins with the collection of the dataset, which typically consists of pairs $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i$ is a feature vector representing the input, and y_i is the corresponding target output. The dataset is commonly partitioned into three disjoint subsets: the training set, used to learn the model parameters by fitting the algorithm to the data; the validation set, used to tune hyperparameters and monitor generalisation during training; the test set, used to evaluate the final performance of the model on unseen data. Then, in *supervised learning* (*i.e.* it uses labelled datasets to train algorithms), the goal is to learn a function $f : \mathcal{X} \to \mathcal{Y}$ that maps input features to output targets. This function is typically parameterised (e.g. by weights in a neural network or coefficients in linear regression) and is learned through optimisation. The algorithm iteratively adjusts the parameters to minimise the discrepancy between the predicted outputs $\hat{y}_i = f(\mathbf{x}_i)$ and the actual outputs y_i in the training data.

In the images below (Fig. 5.1a and 5.1b), the reader can see the resemblance between a biological and an artificial neuron. The biological neuron begins its work with *dendrites*, branching extensions that reach out to gather signals from other neurons. These signals come in the form of chemical or electrical impulses. Similarly, in an artificial neuron, inputs serve the same purpose as dendrites, bringing in information from the dataset or other neurons in the network. Once the signals arrive, they are sent to the *soma*, *i.e.* the cell body. The soma acts as a processing unit, where incoming signals are combined and evaluated. If the cumulative input reaches a certain threshold, the neuron decides to trigger, sending its signal onward. In artificial neurons, this process mirrors the weighted sum of inputs, followed by the application of an *activation function*, which determines whether the neuron activates and passes its signal forward. From the soma, the signal travels down the *axon*, at whose end are the *axon terminals*, which release neurotransmitters to pass the signal to the next neuron. In the artificial equivalent, this

corresponds to the output of the neuron, which is sent to subsequent layers or neurons in the network, forming the next stage of computation.



(a) Biological neuron structure



(b) Artificial neuron (connected to other previous neurons a_i).

Figure 5.1: Biological versus artificial neuron

Despite these similarities, there are some clear differences. Biological neurons rely on complex electrochemical processes and can handle intricate, non-linear dynamics influenced by countless biochemical factors. Artificial neurons, by contrast, are mathematical abstractions. They simplify these processes into weights, sums, and *activation functions*, trading biological complexity for computational efficiency. This simplification is formalized as follows:

Definition 5.2.1. *An artificial neural network is a triplet* (N, V, w) *where* N *is the set of neurons,* V *is the set of connections between neurons (in other words it is* $(i, j)|i, j \in N$*), while* w *is a function* $w : V \to R$ *that defines the weights of the connections.*

The network's structure, resembling its biological counterpart, evolves during training and recall phases to optimize classification performance. Every network is constructed by layers of neurons as shown in Fig. 5.2. In fact, an ANN typically consists of three types of layers: the input layer, which receives the initial data, the hidden layer(s), which perform intermediate computations and feature extraction, and the output layer, which produces the final output.

As mentioned above, each neuron receives input from other neurons, processes it, and then generates an output for the following neuron (*forward propagation*).The connections

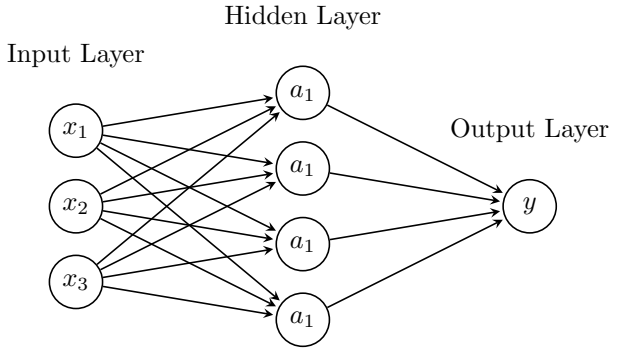


Figure 5.2: Artificial neural network with input, hidden and output layer.

between neurons are weighted, and these weights are adjusted during the training process to improve the network's performance (*back propagation*). Specifically, each neuron computes a weighted sum of its inputs, adds a bias, and applies an activation function to produce an output. Mathematically, the operation of a single neuron can be described in the following steps: weighted sum, activation function and neuron output. The weighted sum is described as:

$$z = \Sigma_i^n w_i x_i + b$$

where:

- x_i are the input features,
- w_i are the weights associated with each input,
- b is the bias term,
- z is the linear combination of inputs and weights.

The result of the weighted sum is given to an activation function $\sigma(z)$ which introduces non-linearity into the model, allowing it to learn complex patterns. The output of the neuron is indeed the result of this function: $a = \sigma(z)$, where a is the output of the neuron after applying the activation function.

The entire processing of data can be divided in *forward propagation*, when the input data is passed through the network layer by layer, and *backpropagation*, when the result of the network goes backward adjusting weights. In the first one, each layer performs the above neuron operations to transform the input into output. For each layer l, is computed:

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}$$
$$a^{(l)} = \sigma(z^{(l)})$$

where:

- $W^{(l)}$ is the weight matrix of layer l,

- $a^{(l-1)}$ is the output from the previous layer (or the input data for the first layer),
- $b^{(l)}$ is the bias vector of layer l,
- $z^{(l)}$ is the linear transformation of the inputs to layer l,
- $a^{(l)}$ is the activation (output) of layer l.

A particular attention must be paid to the activation function. Activation functions are a key component of neural networks. They introduce non-linearity into the network, enabling it to learn and model complex data patterns. Without activation functions, a neural network would behave as a linear model, regardless of the number of layers it has. Choosing the right activation function depends on the specific problem and the architecture of the neural network. By appropriately selecting and applying activation functions, neural networks can achieve better performance and efficiency in learning from data. In general, learning paradigms in machine learning can be divided into *supervised* and *unsupervised* approaches. In supervised learning, the algorithm is trained on labelled data, where each input is associated with a known output, enabling the model to learn explicit input–output mappings. In contrast, unsupervised learning operates on unlabelled data and seeks to uncover hidden patterns, structures, or groupings without predefined targets. In this thesis, we adopt the supervised learning paradigm, as our objective is to train models that can accurately predict and classify outcomes based on labelled examples.

After forward propagation, the backpropagation algorithm begins. It calculates the gradient of the *loss function*, which measures the difference between the predicted output and the true target, with respect to the neural network's parameters. This gradient is used to update the weights and biases of the network to minimize the loss. Backpropagation achieves this by applying the chain rule to efficiently compute the gradients, typically in conjunction with an optimization algorithm. The update rule for gradient descent is:

$$w_i \to w_i - \alpha \frac{\delta L}{\delta w_i}$$
$$b \to b - \alpha \frac{\delta L}{\delta b}$$

where:

- α is the learning rate,
- $\frac{\delta L}{\delta w_i}$ is the gradient of the loss with respect to weight w_i,
- $\frac{\delta L}{\delta b}$ is the gradient of the loss with respect to bias b.

The *loss function*, also known as the *cost* function, is a critical component of training a neural network, similar in importance to the activation function. It quantifies the difference between the network's predicted output and the actual target values, effectively measuring the network's performance. The goal of training is to minimise this loss function, thereby improving the accuracy of the model's predictions.

While the basic concept of ANNs involves interconnected layers of neurons that process and learn from data, various specialized architectures have been developed to address specific types of tasks and data structures more effectively (as the reader will see in Section 5.5 and 5.4).

5.3 Evaluation

Assessing the performance of a neural network is a crucial step in the development and validation of any machine learning model. Indeed, this allows us to determine whether the model is generalising well to unseen data or merely memorising the training examples. This section presents the most common strategies used to evaluate neural networks, including dataset partitioning techniques and performance metrics.

5.3.1 Training, Validation, Test

In *supervised learning*, that is a category of machine learning that uses labeled datasets to train algorithms to predict outcomes and recognize patterns, the quality of a neural network's predictions critically depends on how the data is partitioned during training and evaluation. The standard approach involves dividing the dataset into three disjoint subsets:

- *training* set: this subset is used to fit the model parameters (e.g., weights and biases in a neural network). The learning algorithm iteratively adjusts the model to minimise a predefined loss function on this data. To make it easier, this is the subset used by the model to learn.
- *validation* set: this subset of the data is used during the training phase to guide model selection and to fine-tune hyperparameters, such as the learning rate, the number of layers, or the strength of regularisation. Unlike the training set, the validation set is not used to directly update the model's parameters. Instead, it serves as an independent benchmark to evaluate how well the model generalises to unseen data while it is still being trained. It enables early stopping criteria, where training is halted once the model's performance on the validation set stops improving, and helps detecting overfitting (see Section 5.3.2) by highlighting discrepancies between training and validation accuracy.
- *test* set: this set is reserved for the final evaluation of the trained model. It provides an unbiased estimate of the model's generalisation performance on unseen data. It should only be used once, after all training and model selection processes are complete.

This separation ensures a fair and robust assessment of the model's ability to generalise. In scenarios where data is limited, more sophisticated validation strategies—such as cross-validation—can be employed to make more efficient use of the available samples (discussed in the next section).

5.3.2 K-fold Cross-Validation

When the available dataset is limited in size, as in the case of weather forecasting (where the limitation was due to the scalability of ILASP rather than the dataset itself), using a fixed training/validation/test split may not provide a reliable estimate of a model's generalisation performance. In such cases, K-fold cross-validation offers a more robust alternative by systematically rotating the validation data across different partitions of the dataset.

With this approach, the dataset is randomly partitioned into K equal-sized folds. The model is trained K times, each time using $K-1$ folds for training and the remaining fold for validation. Cross-validation is particularly useful in model selection and hyperparameter tuning, as it enables better utilisation of the available data without needing to sacrifice a large portion for validation. However, it is computationally more expensive than a single train/validation split, since the training process is repeated K times. Common values of K include 5 and 10, balancing computational cost and variance reduction. The reader can find an example of 5-cross fold validation in the Fig. 5.3.

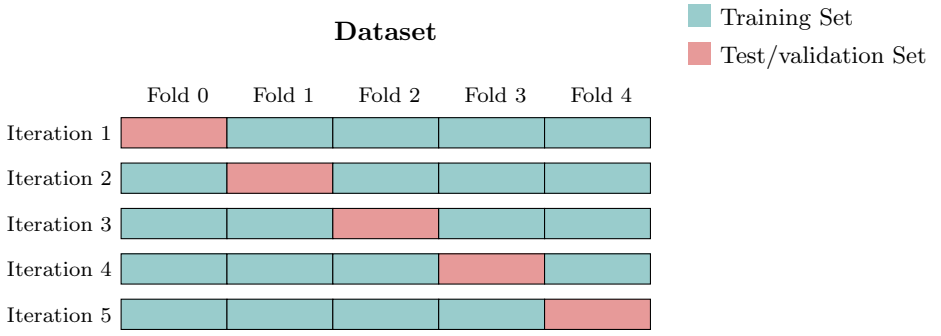


Figure 5.3: K-fold cross-validation (k=5).

The final performance is typically computed as the average of the validation scores across all K folds. This approach reduces variance due to data partitioning and provides a more reliable estimate of how the model is likely to perform on unseen data.

Indeed, despite the impressive performance of neural networks across a wide range of tasks, they can exhibit some issues. The first one is the one of *overfitting*. Overfitting occurs when a neural network learns to memorise the training data rather than generalise from it. This typically leads to excellent performance on training samples but poor results on unseen data. Overfitting is particularly problematic in settings with limited labelled data or high intra-class variability, as is often the case in biological imaging. Regularisation techniques such as *dropout*, *weight decay*, and *early stopping* are commonly employed to mitigate overfitting. In particular, in this thesis, except for K-fold cross-validation, we used an early stopping function. It works as follows:

- During training, the model's loss (or accuracy) is monitored on both the training set and a separate validation set after each epoch.
- If the validation loss starts to increase (or validation accuracy decreases) while the training loss continues to decrease, this typically indicates that the model is beginning to overfit the training data and so the training will be stopped.

Its counterpart is *underfitting*. Underfitting arises when a model is too simple to capture the underlying structure of the data. This results in poor performance on both training and test sets. Underfitting may be caused by insufficient model complexity, inadequate training duration, or small datasets. Balancing model capacity with the complexity of the target task is therefore critical to ensure sufficient learning without overgeneralisation.

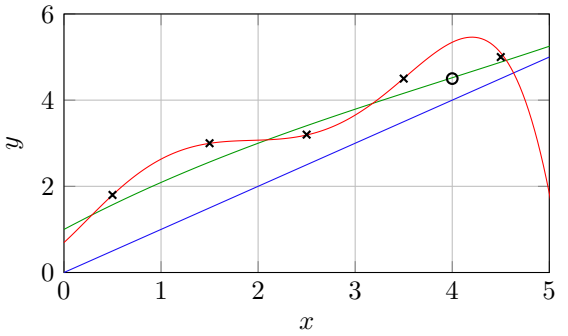


Figure 5.4: Functions learned to approximate the training set (marked by ×), illustrating underfitting (blue), overfitting (red), and a well-generalising curve (green). The circle denotes a data point not used during training.

The curves in Figure 5.4 illustrate the functions learned by the system to perform predictions on unseen data. Consider the case in which we want to predict the value of y for the circle data point: both the red and blue curves will result in higher prediction errors compared to the green curve, which better captures the underlying data distribution and thus generalises more effectively.

5.3.3 Metrics

Once a neural network has been trained and validated, it is essential to quantify its performance using appropriate evaluation metrics. These metrics provide a standardised way to assess how well the model performs. The choice of metric depends on the nature of the task, the structure of the data, and the importance of different types of errors. In classification settings, particularly those involving imbalanced classes, relying solely on overall accuracy can be misleading. Indeed, several standard metrics are used to evaluate model performance. The following are the most common (TP, TN, FP and FN are respectively true positive, true negative, false positive and false negative):

- *accuracy* measures the proportion of correctly predicted instances over the total number of predictions

$$acc = \frac{TP + TN}{TP + TN + FP + FN}$$

- *precision* quantifies the proportion of true positive predictions among all instances predicted as positive, indicating how reliable the model is when it makes a positive prediction

$$Pr = \frac{TP}{TP + FP}$$

- *recall* measures the proportion of true positive instances that were correctly iden-

tified by the model, reflecting its ability to detect relevant cases

$$Rec = \frac{TP}{TP + FN}$$

- *F1-score* is the harmonic mean of precision and recall, offering a balanced metric that is especially useful when false positives and false negatives carry similar costs

$$F1 = \frac{2 \times Pr \times Rec}{Pr + Rec}$$

These metrics are typically computed per class in multi-class settings and can be averaged using *macro*, *micro*, or *weighted* strategies depending on the evaluation goals:

- macro and micro precision

$$Pr_{Macro} = \frac{1}{N} \sum_{i=1}^{N} \frac{\mathrm{TP}_i}{\mathrm{TP}_i + \mathrm{FP}_i} \tag{5.1}$$

$$Pr_{Micro} = \frac{\sum_{i=1}^{N} \mathrm{TP}_i}{\sum_{i=1}^{N} (\mathrm{TP}_i + \mathrm{FP}_i)} \tag{5.2}$$

- macro and micro recall

$$Rec_{Macro} = \frac{1}{N} \sum_{i=1}^{N} \frac{\mathrm{TP}_i}{\mathrm{TP}_i + \mathrm{FN}_i} \tag{5.3}$$

$$Rec_{Micro} = \frac{\sum_{i=1}^{N} \mathrm{TP}_i}{\sum_{i=1}^{N} (\mathrm{TP}_i + \mathrm{FN}_i)} \tag{5.4}$$

where N is the number of classes, TP_i is the number of true positives for class i, FP_i is the number of false positives for class i, and FN_i is the number of false negatives for class i.

5.4 Convolutional and Recurrent Neural Networks

In this section, we provide an overview of three fundamental neural network architectures used in our work, highlighting their main characteristics and typical application domains. All of them were applied to the construction of the neural network for lightning prediction.

5.4.1 Convolutional Neural Network

CNN are particularly well-suited for tasks involving spatial data, such as image recognition, classification, and segmentation, because they are designed to exploit the local structure and spatial correlations present in such data. At their core, CNNs consist

of multiple layers that transform input data through a combination of convolutional, pooling, and fully connected operations, allowing the network to automatically learn hierarchical feature representations. The convolutional layer is the core building block of a CNN. It applies a set of learnable filters (or *kernels*) to the input image, which slide (or convolve) across the width and height of the input volume. This operation produces a feature map (or activation map) that highlights the presence of certain features in the input, such as edges, textures, or patterns. Sometimes, before applying the filters, the input image may be *padded* with additional pixels around the boundary. Padding helps maintain the spatial dimensions of the input image and can be of different types like zero-padding or reflective padding. As each filter traverses the input image, it moves in fixed increments called *strides*, covering one patch (receptive field) at a time. During this process, the filter performs element-wise multiplication with the pixel values within the patch. In particular, the operation the CNN layer is making is the following:

$$(f * x)(i,j) = \sum_m \sum_n x(i+m, j+n) \cdot f(m,n) \tag{5.5}$$

where f is the filter, x is the input image, and (i,j) are the coordinates of the output feature map. Below (Eq. 5.6) there is an example with 2×2 filter. To calculate the first element, the filter is applied to the initial patch of the input. The filter then slides by a distance equal to the stride, and the same operation is performed to calculate the next element. This process is repeated across the entire input, producing the final feature map.

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1n} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2n} \\ x_{31} & x_{32} & x_{33} & \dots & x_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & x_{m3} & \dots & x_{mn} \end{bmatrix} (*+) \begin{bmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \end{bmatrix} \begin{bmatrix} k_{11}x_{11} + f_{12}f_{12} + f_{21}f_{21} + f_{22}f_{22} & \dots \\ \vdots & \ddots \end{bmatrix} \tag{5.6}$$

As shown, the result of a convolution operation is smaller in size than the input image. Given a kernel $k \times k$, a stride s and a padding p, the resulting output dimension $H_{out} \times W_{out}$ for a convolutional layer on an input of dimension $H_{in} \times W_{in}$ is computed as:

$$H_{out} = \left\lfloor \frac{H_{in} - k + 2 \times p}{s} \right\rfloor + 1, \; W_{out} = \left\lfloor \frac{W_{in} - k + 2 \times p}{s} \right\rfloor + 1$$

To maintain the spatial dimensions of the input after convolution, *padding* is introduced. Padding consists of adding additional rows and columns around the borders of the input image, typically filled with zeros, which ensures that the output feature map retains the same height and width as the original input.

Example 5.4.1
Filters play an important role: in this example, the chosen filter is useful to detect edges. Higher pixel values represent the brighter portion of the image and the lower pixel values represent the darker portions.

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} (*+) \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{bmatrix}$$

As we can see, the final matrix highlights the edges present in the original matrix.

After each convolution operation, an activation function is applied to introduce non-linearity into the model. The most commonly used activation function in CNNs is the *Rectified Linear Unit* (ReLU). This function outputs the input directly if it is positive; otherwise, it outputs zero.

$$\text{ReLU}(z) = \max(0, z) \tag{5.7}$$

ReLU and its variants (e.g., *Leaky* ReLU) are generally preferred for hidden layers due to their efficiency and ability to mitigate the *vanishing gradient problem*, *i.e.* when gradients become extremely small during backpropagation.

Pooling layers, typically implemented as *max*-pooling or *average*-pooling, follow convolutional layers to progressively reduce the spatial dimensions of the feature maps. This downsampling operation serves two main purposes: it reduces computational complexity and provides translational invariance, meaning the network becomes less sensitive to the exact position of features within the input. Max pooling is the most common pooling operation, where the maximum value within a window is selected (see Equation 5.8). This also introduces slight translation invariance, ensuring small shifts in the image do not significantly affect the output values.

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{bmatrix} \rightarrow \begin{bmatrix} max(x_{11}, x_{12}, x_{21}, x_{22}) & max(x_{13}, x_{14}, x_{23}, x_{24}) \\ max(x_{31}, x_{32}, x_{41}, x_{42}) & max(x_{33}, x_{34}, x_{43}, x_{44}) \end{bmatrix} \tag{5.8}$$

Example 5.4.2
Maxpooling with 2×2 kernel size, zero stride and zero padding.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 10 & 10 & 0 \\ 0 & 10 & 10 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

5.4.2 Recurrent Neural Network

In the learning and recognition processes occurring in the human brain, time plays a crucial role. For instance, interpreting the meaning of a sentence relies on the complete sequence of words, rather than individual terms. Each word gains meaning relative to the preceding ones, culminating in the understanding of the full sentence. Traditional

neural networks lack this temporal dependency, as input sequences do not influence the output. RNN are a class of artificial neural networks particularly well-suited for sequence modelling tasks. Unlike traditional feedforward neural networks, which process inputs independently, RNNs have an internal memory that allows them to capture temporal dependencies in sequential data. This memory mechanism enables RNNs to effectively process sequences of variable length and to exhibit dynamic behavior over time.

The basic architecture of an RNN layer consists of three main components:

- Input: at each time step t, the RNN layer receives an input vector $x^{(t)}$, representing the input at that time step. This input could be a word in a sentence, a frame in a video, or any other element of the sequence.
- Hidden State: the RNN layer maintains a hidden state vector $h^{(t)}$ that encapsulates the network's memory of past inputs up to time step t. This hidden state serves as the internal representation of the sequence and evolves over time as new inputs are processed.
- Recurrent Connection: at each time step, the RNN computes the hidden state $h^{(t)}$ based on the current input $x^{(t)}$ and the previous hidden state $h^{(t-1)}$. Mathematically, this can be expressed as:

$$h^{(t)} = f(W_{hx}x^{(t)} + W_{hh}h^{(t-1)} + b_h)$$

  where:

  - f is the activation function (e.g., tanh or ReLU),
  - W_{hx} is the weight matrix for the input,
  - W_{hh} is the weight matrix for the recurrent connection,
  - b_h is the bias vector.

- Output: depending on the specific task, the RNN layer may produce an output $y^{(t)}$ at each time step, which could be used for prediction, classification, or any other task. The output is typically computed based on the hidden state $h^{(t)}$ at that time step.

During training, RNNs are trained using *Backpropagation Through Time* (BPTT), which is an extension of the backpropagation algorithm for feedforward neural networks. BPTT involves unfolding the RNN over time into a DAG[1] and computing gradients through the unfolded network. These gradients are then used to update the parameters of the RNN through gradient descent or its variants.

RNNs face several challenges, including vanishing and exploding gradients, difficulty in capturing long-term dependencies, and computational inefficiency. To address these challenges, advanced variants of RNNs, such as *Long Short Term Memory* (LSTM) networks, have been developed. These architectures incorporate sophisticated gating mechanisms that alleviate the vanishing gradient problem and improve the model's ability to capture long-range dependencies.

[1] A directed acyclic graph is a directed graph with no cycles.

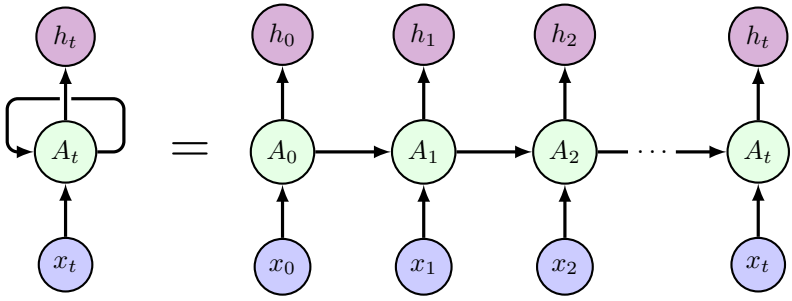


Figure 5.5: Recurrent Neural Network

LSTM network is a specialized RNN introduced by Hochreiter and Schmidhuber in 1997 [77]. LSTMs are designed to retain prior information and utilize it for subsequent outputs, ensuring minimal information loss. Unlike standard RNNs, which struggle with dependencies over long sequences, LSTMs can better capture long-term dependencies, enabling effective learning from distant input elements in a sequence.

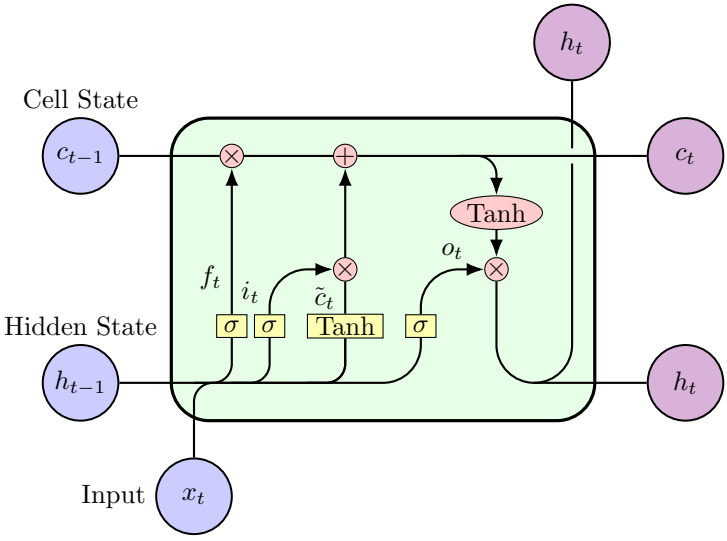


Figure 5.6: LSTM unit

The structure of an LSTM network is based on the modular repetition of neural networks, similar to standard recurrent neural networks. However, unlike RNNs, each LSTM module comprises four interconnected layers rather than just one. The core mechanism involves a linear path for transmitting information, augmented by minor operations (represented as small circles) and neural network processing (rectangles) as shown in Fig. 5.6. LSTMs use gates, which are specialized structures combining a neural network and an operation to manage information flow. Gates control which information is retained or discarded, enabling the network to address long-term dependencies effectively. We can distinguish three gates. The first gate is called *forget gate* and

determines which information of the previous state (c_{t-1}) should be discarded. This decision is made using the previous output, denoted as h_{t-1}, and the current input, x_t, evaluating each element of the cell state, denoted as c_{t-1}. The next step updates the old cell state, determining which new information is needed. This involves two operations which involve *input gate*: the first decides which values to update, while the second creates a vector of new potential value ($\tilde{c}_t$), that could be added to the state. At the end of these two steps, unnecessary information has been removed, and relevant new information has been added. Finally, the last step produces the output. In this step, the part of the cell state to be provided as output is determined. The resulting value is then multiplied by the transformed current cell state, where the transformation is applied using tanh function to ensure the value lies within the range $[-1, 1]$. The operations can be summed up as follows:

- Forget gate: $f_t = \sigma(W_f \dot{[}h_{t-1}, x_t] + b_f)$
- Input gate: $i_t = \sigma(W_i \dot{[}h_{t-1}, x_t] + b_i)$,
  $\tilde{c_t} = tanh(W_c \dot{[}h_{t-1}, x_t] + b_c)$
- Output gate: $o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$
  $h_t = o_t * tanh(c_t)$

The introduction to LSTM described above focuses on scalar or n-dimensional values, but recurrent networks can be extended to handle 2D or 3D inputs. This allows them to be integrated before a convolutional network to enhance its capabilities.

5.4.3 Convolutional Recurrent Neural Network

In a *Convolutional LSTM* (ConvLSTM) layer, the input, output, and hidden states, as well as the update, forget, and output gates, can be organised as three-dimensional tensors, where the first dimension corresponds to time and the remaining two dimensions represent the spatial layout of the data. The ConvLSTM layer computes the hidden state at time step t for each cell in the grid by taking into account both the inputs at the current time step and the hidden states of neighbouring cells from the previous time step $(t-1)$. The operations performed in this layer are governed by the following set of equations:

- $update_t = \sigma(W_{xu}X_t + W_{hu}H_{t-1} + W_{cu}(*+)C_{t-1} + b_u)$
- $forget_t = \sigma(W_{xf}X_t + W_{hf}H_{t-1} + W_{cf}(*+)C_{t-1} + b_f)$
- $C_t = forget_t(*+)C_{t-1} + update_t(*+)tanh(W_{xc}X_t + W_{hc}H_{t-1} + b_c)$
- $output_t = \sigma(W_{xo}X_t + W_{ho}H_{t-1} + W_{co}(*+)C_{t-1} + b_o)$
- $H_t = o_t(*+)tanh(C_t)$

where the products between matrices are performed element-wise (point-wise), while $(*+)$ denotes the convolution operator.

5.5 You Only Look Once

Object detection is a very important task in computer vision, enabling systems to identify and locate multiple objects within an image or video. Unlike image classification, which assigns a single label to an image, object detection combines classification and localization to not only determine the type of objects present but also their precise positions using bounding boxes. Modern object detection models, such as *Region-based CNN* (R-CNN), *Fast* R-CNN, and *Faster* R-CNN, have made important advancements in accuracy and speed by exploiting convolutional neural networks. However, many of these methods rely on multi-stage processes that can be computationally intensive and time-consuming.

YOLO is a state-of-the-art, real-time object detection algorithm introduced in 2015 by Redmon et al. [78] and we decided to use it in the veterinary medicine case. It represents a paradigm shift in object detection by reinventing it as a single regression problem. Instead of employing multiple passes or stages to process an image, YOLO performs object detection in a single forward pass through the neural network. This approach allows YOLO to simultaneously predict bounding box coordinates and classify objects in one single pass. By dividing the input image into a grid and using grid-based and bounding box predictions, YOLO determines whether objects fall within specific cells while predicting their class probabilities.

The real-time processing capability of YOLO is one of its great advantages. It operates at high speeds, making it particularly well-suited for applications that demand low-latency performance. Moreover, YOLO's analysis of the entire image in one pass reduces the likelihood of false positives, as it benefits from greater contextual understanding compared to region-based methods. This combination of efficiency, accuracy, and versatility emphasizes the importance of YOLO in advancing the capabilities of real-time object detection systems.

5.5.1 YOLO algorithm

As mentioned, YOLO directly predicts bounding box coordinates and class probabilities in one pass. Specifically, the YOLO algorithm divides the input image into an $S \times S$ grid and for each of them, the model predicts:

- *bounding boxes*: a fixed number B of bounding boxes, along with confidence scores, that indicate the likelihood of an object being present and the precision of the bounding box, and
- *class probability*: a conditional class probabilities C, representing the likelihood of specific objects within a bounding box, conditioned on the presence of an object.

To manage these predictions, YOLO outputs a high-dimensional tensor containing bounding box coordinates (x, y, w, h) (where x and y define the centre of the object's bounding box, while w and h correspond to its width and height), object confidence scores, and class probabilities. This tensor is then flattened into a vector for efficient processing. The bounding boxes and class probabilities undergo post-processing steps, including *Non-Max Suppression* (NMS), which eliminates redundant overlapping boxes by retaining only the box with the highest confidence score.

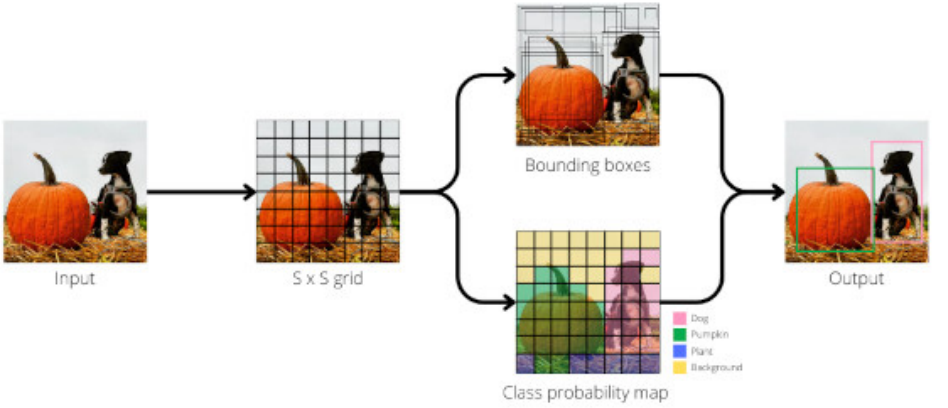


Figure 5.7: YOLO steps

We said that YOLO generates a tensor for each grid cell, containing multiple values for each cell. This vector is often large and complex to handle directly, so to manage the high-dimensionality of the output, the YOLO algorithm employs the *vector generalization*. To simplify processing, the tensor is flattened into a one-dimensional vector. This transformation reduces the dimensionality, making it more manageable for further computation. After the flattening step, the vector is passed through a softmax function, which converts the class scores into probabilities, ensuring that the class predictions for each object are represented as valid probability distributions.

5.5.2 Architecture of YOLOv8

We are now going to discuss the architecture of YOLOv8 since when the project started, that version was the latest. YOLOv8, unlike previous versions, is able to detect multi-scale objects, which means that the model is capable of detecting objects of several sizes. Furthermore, it is in this version that *Exponential Linear Unit* (ELU) activation function was introduced. ELU helps in speeding up learning, not only in YOLO but in deep neural networks in general, by mitigating the vanishing gradient problem and allowing faster convergence. In YOLOv8 the loss is computed by the *Generalized Intersection over Union* (GIoU) loss. GIoU is an enhanced version of the *Intersection over Union* (IoU) metric (explained later in this section), designed to account for the shape and size of bounding boxes. This improvement helps to achieve greater precision in object localization.

The YOLO model is composed of three main components: the backbone, the neck, and the head (the complete architecture is shown in Figure 5.8).

- *Backbone*: it is a series of convolutional layers designed to extract multi-scale features from an input image by preprocessing the image to detect patterns and structures essential for object recognition. Usually, the backbone is pre-trained on a classification dataset at a lower resolution than required for object detection.

This pre-training helps the model learn general features, but finer details are incorporated later during detection-specific training.

- *Neck*: it connects the backbone and the head by using the features extracted by the convolutional layers. It typically includes fully connected layers that enhance these features and produce predictions for object probabilities and bounding box coordinates.

- *Head*: it is the final output layer of the network, responsible for generating the detection results. It is designed to output an $S \times S \times (C + B * 5)$ tensor, where S is the grid size, C is the number of classes, and B is the number of bounding boxes predicted per grid cell.

5.5.3 Evaluation Metrics for YOLO

To assess the performance of object detection models like YOLO, various evaluation metrics such as IoU and *Average Precision* (AP) are used.

Intersection over Union measures the degree of overlap between the predicted bounding box P and the ground truth bounding box G. The IoU is calculated as the ratio of the area of intersection between P and G to the area of their union.

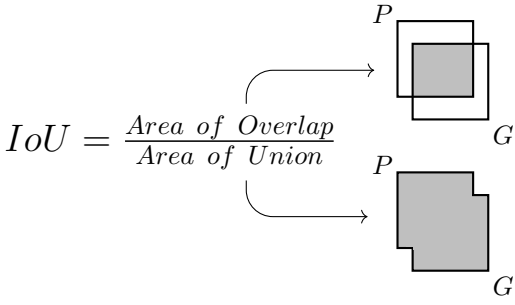


Figure 5.9: Intersection over Union visually explained

The IoU score ranges from 0 to 1, where 0 indicates no overlap and 1 signifies a perfect match between the predicted and ground truth boxes. Indeed, a higher IoU score corresponds to more accurate object detection.

AP encapsulates the precision-recall curve, which is derived by adjusting the detection threshold. Precision represents the fraction of correctly identified objects among all predicted positives, while recall denotes the fraction of actual objects successfully detected. The AP score is obtained by averaging precision values across recall levels ranging from 0 to 1, measuring the model's ability to detect objects at different confidence levels. A higher AP score signifies better detection performance. In practice, *mean Average Precision* (mAP) is often used to aggregate the AP scores across multiple classes. The mAP is calculated by averaging theAP scores for each class, providing a

complete evaluation of the model's performance across all object categories. mAP is followed by a number or an interval of numbers. Specifically, mAP50 employs a fixed IoU threshold of 0.50, meaning a prediction is considered correct if its bounding box overlaps with the ground truth by at least 50%. This metric primarily reflects the model's ability to detect objects under less strict conditions. Conversely, the mAP50-95, calculates the average precision over a range of IoU thresholds, spanning from 0.50 to 0.95 in 0.05 increments. This more stringent metric provides a detailed assessment of both object detection accuracy and localisation precision, making it a more exhaustive measure of model performance.

5.6 Summary

5.6.1 Strengths

- high accuracy
- can work with big amount of data

5.6.2 Limitations

- require large datasets
- black-box
- long time for annotating images

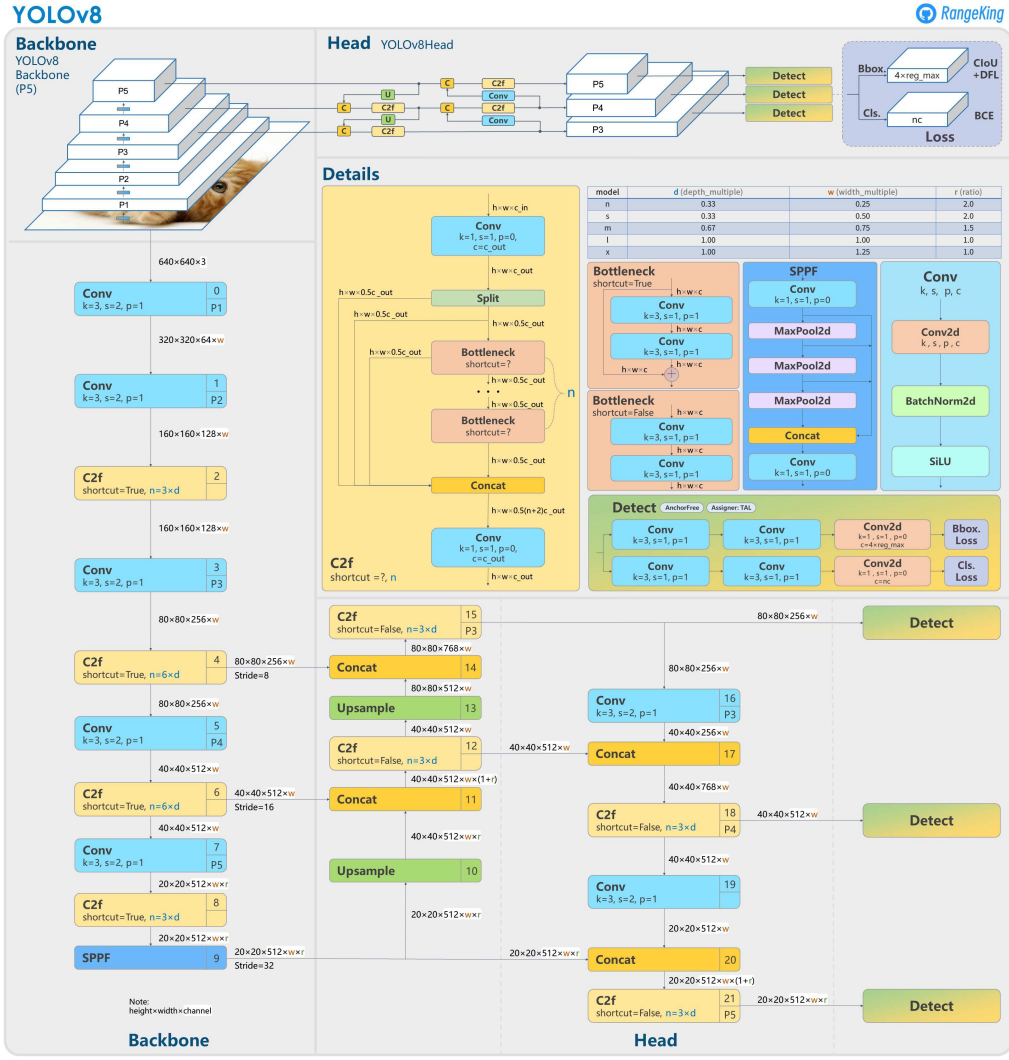


Figure 5.8: YoloV8 architecture

III
Contributions

6

Proposed Methodology

In this chapter, we present the general methodology proposed for integrating deep learning with logic programming. The chapter is structured into three sections, each corresponding to one of the contributions discussed in the subsequent chapters of this part of the thesis. For each case study, a diagram illustrating the corresponding processing pipeline is provided. Components represented with dotted lines indicate elements that are either under development or planned as part of future work. Furthermore, we do not always have the "*Feature extraction*" component as some data can be directly (or at least with a negligible effort) translated into logical language.

6.1 Selection of Case Studies

The three proposed contributions were motivated by different considerations. In particular, the research project was primarily aimed at demonstrating the integration of deep learning and logic programming through inductive logic programming, without a specific focus on a single application domain.

The University of Udine had an established collaboration with OSMER, within which a project was already ongoing. We therefore decided to evaluate the proposed pipeline in this context, as we were already actively working on the deep learning component. At the same time, the Department of Giuridical Science (DISG) of Udine expressed interest in translating the Italian Criminal Code into an automated framework. Given that the structure of legal norms is inherently logic-based, we adopted logic programming for this purpose. Furthermore, DISG indicated an interest in learning rules from previous judicial decisions, which provided an additional motivation to test our approach in this context. Finally, the Department of Agricultural, Food, Environmental and Animal Sciences (DI4A) of the University expressed a similar interest within its domain. In particular, the department requested the development of a neural network for the identification of spermatozoa. Once the network had been successfully developed, we recognised that this task also provided a suitable and meaningful application for evaluating the proposed pipeline.

All the considered domains were well suited to the objectives of the project, as large datasets were already available for each of them (with the exception of the veterinary domain, for which DI4A agreed to construct a dedicated dataset). Moreover, the underlying knowledge in these domains could be readily translated into a logical formalism, and we were able to continuously benefit from the expertise of domain specialists throughout the project. Finally, considering such diverse application domains allows us to demonstrate that the proposed framework operates independently of the specific domain.

6.2 Weather Forecasting

Regarding weather forecasting, we encounter several distinct problems that, nonetheless, share a common methodological foundation.

6.2.1 Explaining Rainfall

The first problem we tackled is described in Section 7.2 and focuses on explaining the predicted rainfall level in a specific region using only data collected from meteorological stations. As this type of data does not allow for long-term forecasting, we restrict our analysis to short-term predictions, using current observations to estimate weather conditions for the following hour. The pipeline is described in Figure 6.1.

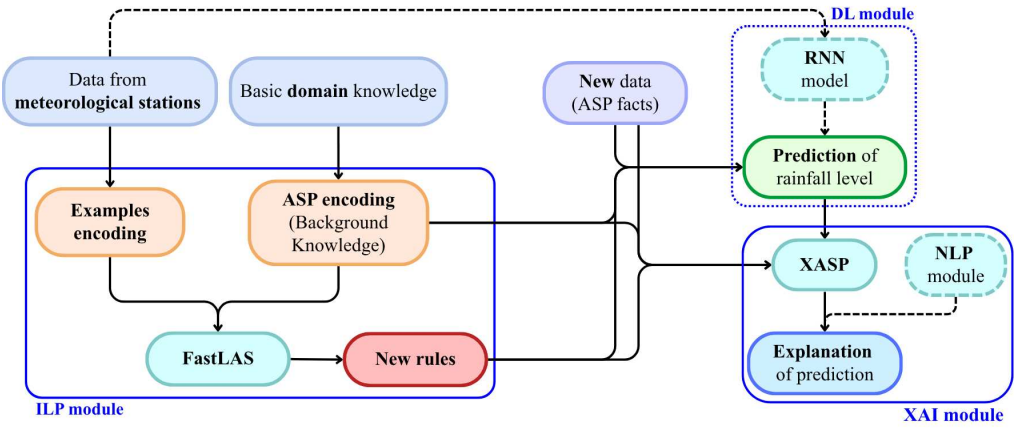


Figure 6.1: Pipeline for explaining rainfall levels.

The first part of the pipeline consists in generating logical rules that will be used for the explanation. The process begins with the collection of meteorological data obtained from weather stations, including variables such as temperature, humidity, and other relevant parameters. These data are recorded with an hourly frequency. The collected measurements are then encoded as FastLAS examples, while general domain knowledge, such as the admissible ranges of each variable, is encoded as background knowledge. Once this encoding phase is complete, FastLAS is executed, producing a set of rules that capture and describe the observed examples.

The second part concerns the explanation of predictions. When new data are recorded from the meteorological station, they are encoded in ASP. Combined with the previously defined background knowledge and the rules learned by FastLAS, this encoding produces a prediction, represented by an answer set containing the corresponding outcome. This answer set is then processed by xASP to generate a graphical explanation of the reasoning that led to the prediction.

In the future, two additional components could be integrated into the system: the first, a RNN-based model to enhance predictive capabilities; and the second, an *Natural Language Processing* (NLP) module to automatically generate textual explanations tuned with examples from human explanations drawn from official reports.

6.2.2 Modeling Advection

The second problem concerns the modelling of the advection of meteorological variables using logic programming, and the use of the resulting rules to explain specific weather events. The overall process is illustrated in Figure 6.2.

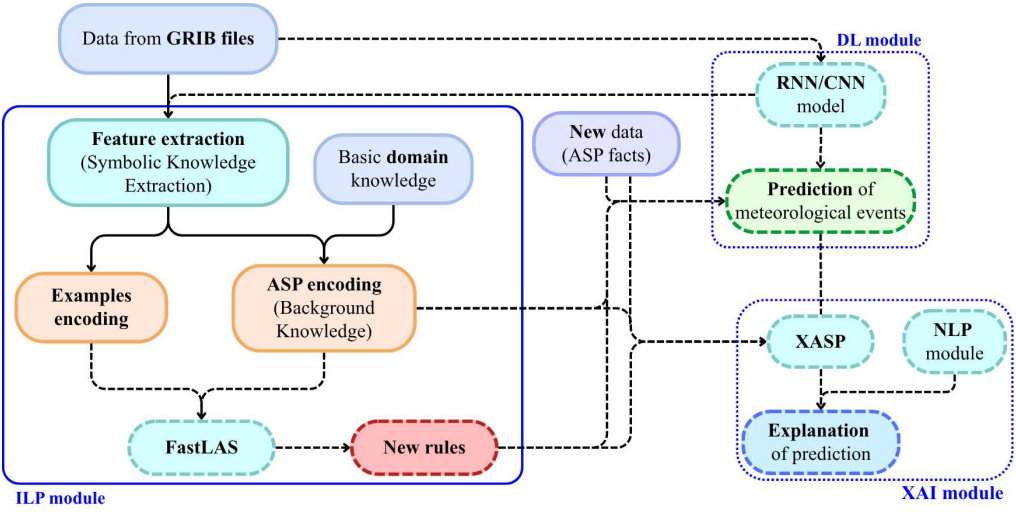


Figure 6.2: Pipeline for advection

The first step involves extracting a set of features from the GRIB files through a series of preprocessing operations. The objective is to derive information about the position and dimensions of atmospheric masses, as well as to reconstruct the trajectory they followed to reach their current location. Those features are then encoded in ASP, to construct the background knowledge, and in examples suitable for FastLAS.

In future work, we plan to employ FastLAS to learn the rules governing the movement of atmospheric masses and the occurrence of meteorological events, with the goal of using these rules to generate more tailored explanatory models. Additionally, a RNN/CNN-based module might be integrated to enhance the feature extraction process, particularly for the tracking and detection of atmospheric masses. This module will be developed only if the existing tobac implementation will be found to be insufficient. Finally,

an NLP component will be developed to improve the readability of the explanations produced by xASP.

It is important to note that our objective is not to provide a complete scientific explanation of the underlying causes of meteorological events. Rather, we aim to generate explanations that resemble the kind of descriptive accounts typically provided by meteorologists when communicating with the public.

6.2.3 Lightning Prediction

The last problem we considered is analysed in Section 7.2. Referring to Figure 6.3, both the logical and explainability components are currently missing. At this stage, the system includes a CNN/RNN-based model that predicts lightning occurrences using a wider and more complex set of meteorological variables, as described in Section 7.4.1. Once the deep learning module has been further improved, the goal is to extract features from the dataset that capture not only instantaneous system states, as was done in the first problem, but also higher-level, more abstract characteristics. This is particularly important in this case, given the large volume and complexity of the available data.

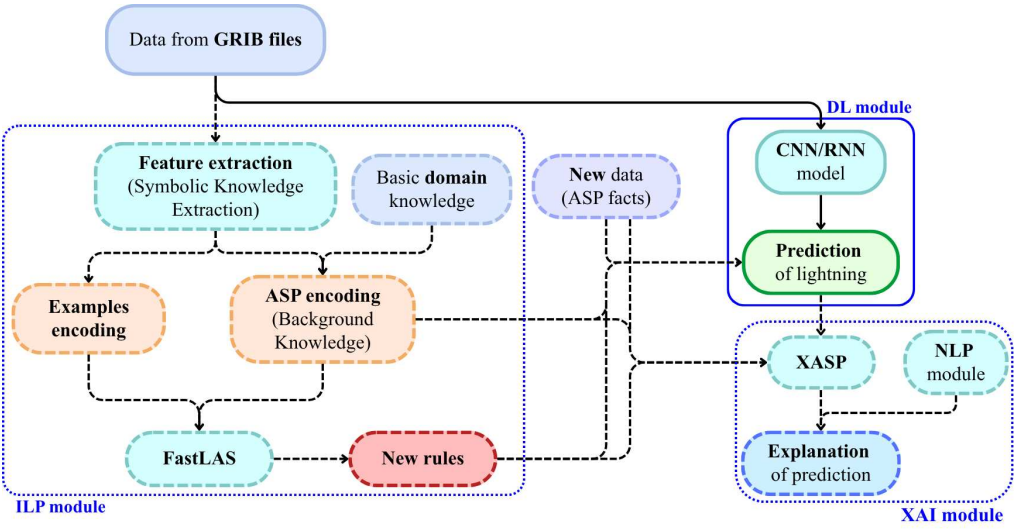


Figure 6.3: Pipeline for lightning prediction.

6.3 Juridical Reasoning

In the juridical field the idea is to learn from the judgment of the Court of Cassation and then use this knowledge to explain new judgment. In this case the aim is to explain something that has already been decided by a judge. The pipeline is described in Figure 6.4.

As in the other cases, the first component of the pipeline is the ILP module. In this context, the Articles of the Italian Criminal Code are encoded as background knowledge,

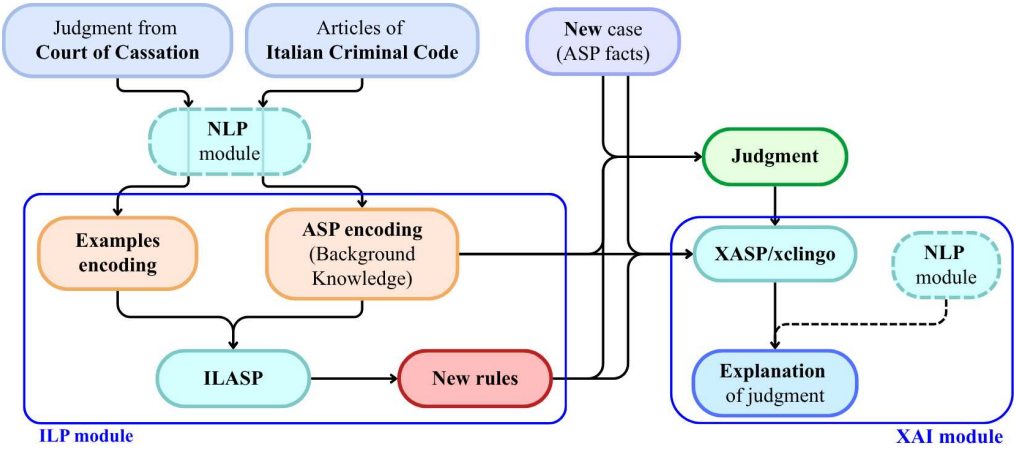


Figure 6.4: Pipeline for juridical explainability

while the judgments of the Court of Cassation are used as examples. At present, an NLP-based system capable of automatically translating juridical language into ASP has not yet been developed; consequently, the encoding process is currently performed manually.

Once the new set of rules describing the examples has been learned, it can be used to perform deductions on previously unseen cases. The background knowledge, together with the learned rules, is employed to produce a deduction that should ideally correspond to the judgment of the court. It is important to note that the newly learned rules are annotated to indicate whether a decision relies on previous judgments, since in the Italian legal system, the Articles of the Criminal Code and the rulings of the Court of Cassation do not hold the same normative weight.

The resulting answer set, representing the judgment of the new case, is then processed by xASP and/or xclingo to generate both visual and textual explanations. In the future, an additional NLP-based component will be developed to further enhance the clarity and accessibility of these explanations.

6.4 Veterinary Medicine

The last case is the most complete. The idea is to explain the anomalies of bull's sperm using the combination of YOLO and logic programming.

The first stage involves training a YOLO network to detect and classify spermatozoa in microscopy images. In parallel, symbolic knowledge extraction is performed on the same images by processing them to identify the morphological features of each spermatozoon. This information is then encoded in two forms: as background knowledge, representing the admissible ranges of the relevant variables, and as examples, with one example corresponding to each detected spermatozoon. These encodings are then used by FastLAS to induce new rules describing the observed morphological patterns.

To determine whether a newly observed spermatozoon is anomalous, its correspond-

ing image is first processed by YOLO. The output of the network is then used to extract the morphological features, following the same procedure adopted for encoding the training examples. Unlike the previous problems, in this case the feature extraction relies on the output of YOLO rather than on the original image, since several features depend directly on the predicted bounding box (for instance, the aspect ratio of the spermatozoon). The bounding boxes, which in the training phase were provided by the dataset annotations, are also essential for defining the region of interest. In fact, image processing operations, such as determining the contour of the sperm head, perform more reliably when applied to images containing a single spermatozoon.

Once feature extraction and encoding are completed, an ASP program is constructed using the newly learned rules. The resulting answer set, representing the classification outcome, is then processed by xASP to generate a textual explanation of the reasoning behind the classification. As for the juridical case, an NLP-based component could be added to improve the textual explanation. Until now, the graph produced from xASP is automatically translated in natural language without the need of a deep learning module since the possibilities are few.

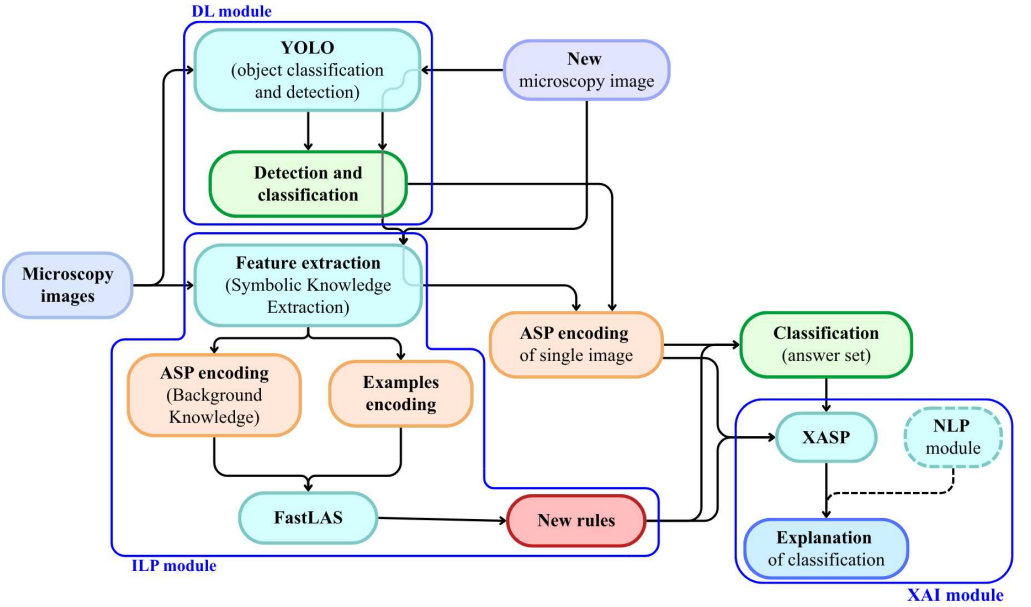


Figure 6.5: Pipeline for anomalies explanations.

6.5 Summary

6.6 Pro and Cons

In this section, we summarise the advantages and disadvantages of the methodology described above. In particular, we analyse the strengths and limitations of each technology

Case study		**ILP module** *background* *examples*	**DL module**	**XAI module**
Weather	Lightnings	FastLAS not developed yet not developed yet	Conv recurrent nn	NLP (not developed yet)
	Rainfall	FastLAS domain specific domain specific	RNN (not developed yet)	XASP + NLP (not developed yet)
	Advenction	FastLAS domain specific domain specific	not DL (tobac)	NLP (not developed yet)
Juridical		ILASP domain specific domain specific	NLP (not developed yet)	XASP + xclingo
Veterinary medicine		FastLAS domain specific domain specific	YOLO	XASP

Table 6.1: Summary of different cases study

employed.

6.6.1 Strengths

We begin by outlining the main strengths of the proposed methodology:

- LP provides intrinsically interpretable and formally grounded reasoning mechanisms, enabling transparent decision-making through declarative rules with well-defined semantics, verifiability, and support for constraint-based modelling.
- ILP extends LP's advantages by allowing symbolic, human-readable hypotheses to be learned directly from data, thereby reducing manual knowledge engineering while preserving strong explanatory power and alignment with domain expertise. It can also handle noisy data and supports user-defined score function to better guide the learning process.
- RCNNs combine spatial feature extraction with sequential modelling, making them particularly effective for structured visual data where contextual or temporal dependencies are relevant.
- Long Short-Term Memory networks further enhance sequential reasoning by capturing long-range temporal dependencies and mitigating the vanishing gradient problem, enabling robust modelling of complex temporal dynamics.
- YOLO offers state-of-the-art performance for real-time object detection by framing detection as a single end-to-end regression problem, achieving high accuracy and computational efficiency while scaling effectively to large datasets.
- explanation frameworks such as xASP and `xclingo` build upon ASP to provide faithful, formal, and structured explanations of both conclusions and intermediate reasoning steps, thereby enabling justification, debugging, and validation of complex hybrid neuro-symbolic systems.

6.6.2 Limitations

Despite their strengths, these approaches also present notable limitations:

- LP systems often suffer from limited scalability and are not suitable for data with noise or uncertainty, as their symbolic nature assumes precise and complete representations.
- ILP faces significant computational challenges due to the combinatorial growth of the hypothesis space and its dependence on the quality and completeness of background knowledge which must be done with the help of domain experts.
- RCNNs introduce increased architectural complexity and training cost, and their learned representations remain largely opaque, making faithful interpretation difficult.

- LSTM networks are computationally expensive, difficult to tune, and prone to overfitting on limited data, while their internal memory mechanisms lack transparent semantic meaning.
- YOLO, despite its efficiency, provides limited insight into the causal factors underlying its predictions, as spatial saliency or bounding-box confidence scores do not constitute high-level explanations.
- Explanation frameworks such as XASP and xclingo, while offering formal and faithful justifications, may incur additional computational overhead and require careful modelling choices, and their explanations can become complex or difficult to interpret for non-expert users when applied to large-scale.
- Interpretable models may underperform compared to complex black-box models and explanations can become too complex to be useful for non-expert users.
- There no exists a universally accepted metrics for explanation quality so it is difficult to make prove the accuracy of a model or a comparison between models.

7

Weather Forecasting

The origins of weather forecasting trace back to the 19^{th} century. Notably, in 1922, Richardson published "*Weather Prediction by Numerical Process*" [79], laying the foundation for modern *Numerical Weather Prediction* (NWP). Weather prediction is of vital importance because of its impact on sectors such as agriculture, construction, road maintenance, and others. Developing automated systems that infer weather patterns using data from meteorological stations is highly beneficial. Today, forecasting techniques have advanced importantly, incorporating sophisticated methods, including machine learning, to improve prediction accuracy and reliability.

In this thesis, we address three distinct challenges in weather prediction. The first problem focuses on explaining rainfall forecasts (Section 7.2), which we approach using solely ILP, the second concerns modelling advection of atmospheric variables to explain weather forecasting (Section 7.3), and the third addresses lightning prediction (Section 7.4), for which we employ RNNs.

7.1 Related Works

Weather prediction has long been a critical area of research. Traditional methods, such as NWP [80], rely on mathematical models that simulate fluid dynamics. These equations, coded into programs, rely on governing laws of physics, numerical methods, and parametrizations of physical processes. While these models are effective, they are computationally expensive and often struggle with capturing fine-grained patterns, especially in complex terrains or short-term forecasts. Other numerical weather prediction models [81], are often complemented by statistical or machine learning methods [82, 83, 84]. Indeed, in recent years, AI techniques have emerged as powerful alternatives or complements to traditional methods. Machine learning models, particularly deep learning architectures, have demonstrated the ability to analyse large volumes of meteorological data, uncover hidden patterns, and produce accurate forecasts. For example, CNNs have been employed to process satellite imagery [85], while RNNs and their variants, such as LSTM networks, have been used to model temporal dependen-

cies in weather data [86]. Other direct applications of AI in weather field are the two of Weyn et. al [87, 88], in which they studied again the application of CNN, and the one of Li et. al [89] which, despite the others, use generative techniques to produce the prediction.

Beyond prediction, dedicated frameworks have been developed to analyse and track atmospheric features directly from observational or simulated data. One such example is *Tracking and Object-Based Analysis of Clouds* (tobac) [90, 91], a Python-based package designed to identify, track, and characterise coherent structures such as clouds, convective cells, and air masses. The framework has demonstrated robust performance and versatility across several research projects and applications [92, 93, 94, 95].

While these models can achieve impressive accuracy, interpreting the inner workings of complex systems, such as neural networks, remains a relevant challenge. To address this, explainable AI techniques are being developed to clarify how predictions are made. For instance, Labe et al. [96] applied XAI to ANNs to analyse historical data and identify factors driving the rise in summer temperatures in the United States. Similarly, other studies, such as [97], have explored XAI strategies for weather forecasting.

7.2 Explaining Rainfall

This section focuses on the study we conducted to explain prediction of rain levels and is an extension of the work presented in [98]. In particular we learned rules from FastLAS and then used them for prediction and explanation. In addition, we compare the results obtained with FastLAS against the performance of other machine learning algorithms, namely *Support Vector Machine* (SVM), *Random Forest*, and *Decision Tree*.

7.2.1 Dataset

The dataset used to train the ILP system consists of data collected from the meteorological station of Udine. Those data are publicly available at `https://www.osmer.fvg.it/archivio.php?ln=&p=dati`. They range over six months and have been divided in two main periods (January, February and March 2024 and May, June and July 2024) and recorded hourly.

Features

Atmospheric variables are fundamental components of Earth's atmosphere that play fundamental roles in shaping weather patterns and influencing various natural processes. These variables include temperature, precipitation, air pressure, wind speed, wind direction (in degrees relative to the North direction), and humidity. Understanding the meaning of their values is crucial for comprehending the dynamics of weather systems and climate patterns. For this reason, we give a brief description of each of them:

- Temperature reflects the degree of heat present in the atmosphere and deeply influences weather phenomena. It affects the rate of evaporation, cloud formation, and atmospheric stability, while shaping local and global climate patterns.

- Precipitation, which in our case includes just rainfalls, represents the deposition of humidity from the atmosphere to the Earth's surface. The distribution and intensity of precipitation events can vary very quickly, both spatially and temporally.
- Atmospheric pressure indicates the force exerted by the weight of air molecules in the atmosphere. Changes in air pressure drive atmospheric circulation, leading to the formation of winds and weather fronts. Understanding air pressure patterns is crucial for predicting the movement of weather systems.
- Wind speed and direction characterise the motion of air masses within the atmosphere. Winds are driven by differences in air pressure and temperature, and they play a critical role in redistributing heat and humidity. Wind patterns influence weather phenomena such as storms, hurricanes, and monsoons.
- Humidity refers to the amount of water present in the atmosphere relative to the maximum amount that the air can hold at a given temperature. It has a key role in the formation of clouds, fog, and dew, and it influences atmospheric stability and the rate of evaporation and transpiration.

In table 7.1, the dataset composition is described: it comprehends 2186 samples, with just a few missing values. The training process utilised a 10-fold cross-validation approach (see Section 5.3.2). In our study, each fold consisted of four days for training, while the remaining days were used for validation. This approach deviates from the common practice, where the training set is typically larger than the test set, but, even if four days may seem insufficient for a training set, the key advantage of ILP systems is indeed their ability to learn effectively from a limited number of examples. The rationale behind this decision is twofold: i) FastLAS lacks scalability, and increasing the number of training days resulted in memory exhaustion; ii) using a very small test set, such as a single day, would not provide sufficient variability, making the evaluation less reliable. However, due to the limited data with high precipitation levels, it was not possible to ensure complete separation between the training sets of different folds. As a result, the training sets of different folds could overlap by up to one day.

Each variable has a different range but, as we can see in 7.1), there is a wide range of values for humidity and pressure in the samples. Atmospheric events such as cold or warm fronts, tropical cyclones, and high or low-pressure systems can rapidly influence atmospheric pressure and humidity, therefore contributing to the high variance. The table also provides other statistical summary, including extremes, mean, σ, and a measure of dispersion (Coefficient of Variation, CV $= \sigma/\mu$). From January to March, the data reflects colder and more stable conditions, with low average rainfall (0.23 mm) and high mean humidity (79.49%). In contrast, from April to June, there was a shift to warmer and more variable weather. While average rainfall stayed low, extremes increased, with a maximum of 45.7 mm, and the dispersion also rose (6.9 vs. 4.2). Humidity decreased slightly, wind speeds increased, and pressure remained mostly stable, with a small decline in the mean value.

7.2.2 Variables domain

The variables considered can span a wide range of values. However, due to scalability issues, managing a large number of possibilities becomes impractical. Therefore, we

	Samples	Missing samples	Max value	Min value	Mean value	Standard deviation	Coefficient of Variation
Jan–Mar					S_{jfm}		
Rain mm	2185	1	17.1	0.0	0.23	0.96	4.23
Temperature °C	2174	12	20.6	-5.0	7.81	4.62	0.59
Humidity %	2185	1	97.0	23.0	79.49	15.03	0.19
Wind speed km/h	2185	1	26.0	1.0	6.03	3.73	0.62
Pressure hPa	2185	1	1026.0	978.0	1004.05	10.17	0.01
Apr–Jun					S_{amj}		
Rain mm	2184	1	45.7	0.0	0.23	1.61	6.89
Temperature °C	2161	24	32.1	1.0	17.55	5.67	0.32
Humidity %	2181	1	99.0	27.0	73.75	17.47	0.23
Wind speed km/h	2173	12	34.0	1.0	7.37	4.23	0.57
Pressure hPa	2182	3	1018.0	987.0	1002.59	0.57	0.01

Table 7.1: Dataset summary

opted to reshape the domain of these variables to ensure that the grounding process does not consume excessive memory resources.

The considerations we made are based on the document "The Climate of Friuli Venezia Giulia" [99] produced by ARPA FVG. It provides basic knowledge about the region's climate by examining key meteorological and climatic variables: precipitation (rain, snow, and hail), temperature, solar radiation, sky conditions, and wind. It comprehends years from 1991 to 2020. This time-frame serves as the reference period for computing climatological averages and it is used for analyses aimed at operational services and decision-making processes for the immediate future in climate-sensitive sectors. In particular, variables were reshaped as follows:

- Rain data were measured in millimetres per day and, to simplify analysis, they have been categorised into four levels: *none* (0 mm), *level_1* (between 0 mm and 2 mm), *level_2* (between 2 mm and 6 mm) and *level_3* (above 6 mm).
- For the temperature, we normalised the values to a range between 0 and 100, using –10 °C as the minimum and 30 °C as the maximum, since these were the observed extremes during the months under consideration. However, in the region, absolute temperature extremes range from approximately –14 °C to 40 °C; therefore, extending the model to cover a wider timespan would require an adjustment of this normalisation.
- Wind speed data did not have extreme cases, so we kept the original values for this variable. During the considered period, wind speed ranged from 0 to 160 km/h, although values exceeding 100 km/h were observed only rarely.
- The wind direction was represented as degrees measured clockwise from the north. We reshaped the domain as follows: N (north), S (south), W (west), E (east), NE (north-east), SE (south-east), SW (south-west), NW (north-west), NNW (north-north-west), and all the other combinations.
- Humidity was already given as a percentage, so we kept the original values that range between 0 and 100.
- Regarding pressure, we opted to normalize them between 800 hPa (the minimum) and 1130 hPa (the maximum), so pressure values are scaled uniformly between 0 and 100.

7.2.3 Methods

The main goal of this work was to create a diagnostic system that can explain the rainfall levels that have occurred. However, the system also predicts the rainfall for the next hour, based on the conditions observed in the current and preceding hours.

Background Knowledge

As mentioned, ILP systems are learning tasks which need a background knowledge B, some examples E and an hypothesis space S_M. Starting from B, it includes a predicate for each variable, such as `temperature(T,X)`, meaning that at time `T` the temperature

was of `X` degrees. Apart from that, we defined a few auxiliary predicates. The most important is the predicate `previous(T1,T2)` that is used when the timestamp `T1` is one hour earlier than `T2`. This predicate was useful to define other characteristics, in particular we introduced predicates to describe the increase or change, between two timestamps, of a variable. For example:

```
increased_temperature(T1, T2) :- previous(T1, T2), temperature(T1, X1),
                                 temperature(T2, X2), X1 < X2.
```

Unfortunately, we had to further simplify the background knowledge to enable the learning of rules that also include conditions. In particular, we use the term FastLAS* to denote the version of FastLAS capable of learning binary comparisons between numerical variables. Indeed, the complexity, due to the high number of variables, caused scalability issues. To address this, we removed the timestamp argument from all predicates by introducing: `current_temperature(X)` predicate, to represent the temperature at the most recent timestamp (that in the restricted case of an example is the current one), and `previous_temperature(X)` predicate, to represent the temperature at the preceding one. Since, as we will see, the context of the examples includes only two consecutive timestamps, the rules learned will inherently refer to these two timestamps as well. In practice, `current_temperature(X)` is defined such that there exists a `temperature(T2, _)` with a timestamp immediately before that of `temperature(T1, X)`; conversely, for `previous_temperature(X)` there must exist a `temperature(T2, _)` whose timestamp is after the one of `temperature(T1, X)`. Once the learning process is completed, it is necessary to remap the predicates to reintroduce the relevant timestamps. The code snippet below illustrates the predicates without timestamps, demonstrating this with just the example of temperature for simplicity.

```
current_temperature(X) :- temperature(T1,X), temperature(T2,_), T1 > T2.
previous_temperature(X) :- temperature(T1,X), temperature(T2,_),T1 < T2.
increased_temperature :- current_temperature(X), previous_temperature(Y), X<Y.
```

Examples

The actual encoding of the dataset takes place within the examples: each example represents a single hour of a day. Since the objective is to learn rules for predicting rain levels, each inclusion contains only the `rain` predicate, while the context includes all other variables (if present) at the same timestamp and one hour earlier. This ensures that the rules learned are related to consecutive timestamps. For instance, if the rule learned is:

```
rain(V0,"none") :- previous_temperature(V_0_deg_temperature),
                   V_0_deg_temperature <= 32, time(V0).
```

then we can translate it into:

```
rain(V0,"none") :- previous_temperature(V0, V_0_deg_temperature),
                   V_0_deg_temperature <= 32, time(V0).
```

where `previous_temperature`, that now has also a timestamp argument, is defined as

```
previous_temperature(T2, X) :- temperature(T1,X), temperature(T2,_),
                               previous(T1,T2).
```

Furthermore, the `rain` value from the previous timestamp is included in the context to enhance consistency during the learning process. We report one example for clarity:

```
#pos(id169@2, {
        rain(9162000, "none")
    },{
        rain(9162000, "level_1"),
        rain(9162000, "level_2"),
        rain(9162000, "level_3")
    },{
        temperature(9162000, 61).     wind_speed(9162000, 9).
        wind_direction(9162000, "E"). humidity(9162000, 4).
        pressure(9162000, 57).
        temperature(9158400, 62).     wind_speed(9158400, 6).
        wind_direction(9158400, "E"). humidity(9158400, 4).
        pressure(9158400, 58).        rain(9158400, "none").
    }).
```

When FastLAS searches for a solution, it aims to minimise the uncovered examples by reducing the sum of their assigned weights. These weights typically serve as indicators of confidence in the examples. In other words, if an example is less representative of a typical case or is potentially noisy, it should be assigned a higher weight, and *vice versa*.

In this work, weights were used to guide the solver to ensure coverage of all rain levels. Specifically, the weights were assigned such that the sum of the weights for examples at each rain level was equal. This approach guarantees that even if some categories have fewer examples, their overall importance remains the same.

It is important to note that all examples in the dataset are positive, and thus, we require every example to be covered by at least one answer set, consistent with the principle of brave induction/reasoning.

Hypothesis Space and Bias

The hypothesis search space was designed so that candidate rules reference the present to make predictions about the future. To achieve this, the body of the rule favours predicates like `increased_temperature` (which inherently refer to two consecutive timestamps) rather than simpler predicates such as `temperature` (which corresponds to a single timestamp). Conversely, the head of the rule exclusively contains the `rain` predicate, as predicting rainfall is the primary objective.

In the rule head, the `rain` predicate includes a constant (e.g., `level_0, ..., level_3`) as its argument, representing the rain level, and a variable to represent the timestamp. Unfortunately, FastLAS operates on a non-recursive OPL task, which means the `rain` predicate cannot appear in the body of learned rules. However, it is allowed in the background knowledge's rules. This limitation is impactful, as it is reasonable to expect that the future rain level is influenced by the current one.

The mode declarations look like the following:

```
#modeh(rain(var(time), const(mm_rain))).
#modeb(previous_temperature(num_var(deg_temperature))).
#modeb(temperature(var(deg_temperature))).
...
```

As mentioned earlier, we aim to prioritise certain predicates over others. Mode declarations alone cannot achieve this prioritisation, so the solution lies in the cost function. The cost function allows us to impose a penalty on rules when specific user-defined conditions are met. For this study, the cost function was designed such that predicates referring to a single timestamp received a penalty of 10. After extensive testing, we observed that the predicates `wind_direction` and `humidity` frequently appeared, indicating their potential importance in predicting rain. To account for this, we assigned them a lower penalty of 5, enabling us to cover more examples. The only negative penalty was assigned to the `previous` predicate because it can influence all predicates related to the present, effectively enforcing references to a prior timestamp.

```
#bias("penalty(5, in_head)  :- head(_).").
#bias("penalty(10, body(X)) :- in_body(X), X = wind_speed(_,_).").
#bias("penalty(10, body(X)) :- in_body(X), X = temperature(_,_).").
#bias("penalty(5, body(X))  :- in_body(X), X = wind_direction(_,_).").
#bias("penalty(10, body(X)) :- in_body(X), X = pressure(_,_).").
#bias("penalty(5, body(X))  :- in_body(X), X = humidity(_,_).").
#bias("penalty(-1, body(X)) :- in_body(X), X = previous(_,_).").
```

This cost function was employed in the first experiment (using FastLAS). Conversely, for the second encoding (using FastLAS*), the scoring function was simplified due to the nature of the reduced variable set and encoding. In this case, a uniform penalty was applied to all predicates without distinction.

```
#bias("penalty(1, head(X)) :- in_head(X).").
#bias("penalty(1, body(X)) :- in_body(X).").
```

7.2.4 Results

FastLAS allowed us to derive meaningful rules for weather explanation. However, these rules inherently lack the precision of neural networks. This limitation arises because rules, by their nature, are more general and rigid compared to a continuous function.

By "rigid", we mean that the evaluation of a rule results in a binary outcome: it is either true or false. For example, a rule may predict a rain level of 2 or fail to do so; there is no possibility of an intermediate result between levels 2 and 3. In contrast, a neural network trained to predict an integer can produce outputs that smoothly transition between categories, allowing for greater flexibility and precision. Furthermore, the reshaping required for this approach, specifically, categorising rain levels into four classes instead of using their exact values, further reduced precision. For instance, predicting a rain level of 2 means the actual value could fall anywhere between 2 mm and 6 mm. This limitation could be resolved in the future by developing a more scalable system capable of handling bigger variable's domain.

Considering this, we now present the results obtained from the two versions of FastLAS, comparing them with the performance of *Support Vector Machine*, *Random Forest*, and *Decision Tree* machine learning algorithms. Starting with the basic version, we demonstrated the feasibility of our approach while also providing valuable insights into weather patterns. Below, we present two representative rules inferred by the system, which illustrate the interactions between various weather factors:

```
rain(V0,"level_1") :- not increased_temperature(V0),  V_0_hPa >= 55,
                      V_0_hPa <= 56, previous_pressure(V0, V_0_hPa), time(V0).
rain(V0,"level_2") :- previous_temperature(V0, V_0_deg_temperature),
                      previous_pressure(V0, V_0_hPa), V_0_hPa <= 55,
                      V_0_deg_temperature >= 57, time(V0).
```

The first rule, for example, which also highlights FastLAS's ability to handle negation effectively, predicts a rain level of 1 under the following conditions:

- temperature has not increased at the current timestamp,
- pressure at the previous timestamp lies within a narrow range of 55 to 56 hPa.

For simplicity, we have presented only two rules, but the model typically selected between 10 and 25 rules per fold. To evaluate the model's performance, we used metrics categorised into macro-averaging and micro-averaging, calculating precision and recall values for both (see in Section 5.3.3 the Equations 5.1, 5.2, 5.3 and 5.4). We also calculated accuracy and *Peirce Skill Score* (PSS) [100]. PSS (Equation 7.1) is a statistical metric used to evaluate the performance of a classification or prediction system, such as weather forecasting. It ranges from -1 to 1, with 1 representing a perfect forecast and 0 indicating no predictive skill (equivalent to random chance). Negative values suggest the predictions are inversely related to the expected outcomes.

$$PSS = \frac{TP}{TP + FN} - \frac{FP}{FP + TN} \tag{7.1}$$

In our results, the PSS is relatively low but still above 0, indicating some predictive capability. For the set S_{jfm}, the PSS is notably lower, likely due to the scarcity of rainfall events or the broad generalisations of the learned rules. Additionally, the FastLAS model achieved a slightly higher PSS compared to FastLAS*, which can be attributed to a more specific scoring function. Table 7.2, includes all these metrics and also mean accuracy and weighted F1.

Table 7.2: Summary of means of performance metrics

	Precision	Recall	Precision	Recall	Accuracy	PSS	Weighted F1
Jan–Mar				Set_{jfm}			
FastLAS	**0.25**	**0.45**	**0.54**	**0.91**	**0.91**	**0.14**	**0.80**
FastLAS*	**0.30**	**0.35**	**0.66**	**0.70**	**0.70**	**0.11**	**0.71**
SVM	0.38	0.34	0.82	0.82	0.82	-0.16	0.79
RandomForest	0.55	0.44	0.81	0.81	0.80	0.44	0.79
Decision Tree	0.45	0.44	0.74	0.74	0.74	0.29	0.76
Apr–Jun				Set_{amj}			
FastLAS	**0.27**	**0.36**	**0.67**	**0.76**	**0.75**	**0.76**	**0.10**
FastLAS*	**0.25**	**0.35**	**0.61**	**0.69**	**0.69**	**0.68**	**0.03**
SVM	0.33	0.32	0.87	0.87	0.87	0.82	0.07
RandomForest	0.38	0.32	0.85	0.85	0.83	0.82	0.14
Decision Tree	0.35	0.34	0.77	0.77	0.77	0.78	0.15

Due to the limited amount of high rainfall data, FastLAS tends to perform better when predicting no rain, even though, as mentioned earlier, we applied a weighting scheme that equalised the total weight for each rain level. That said, the overall accuracy for the S_{jfm} set remains consistently above 0.85. On the other hand, the results from FastLAS* runs, which also use rules with conditions, show lower accuracy. However, while accuracy is lower, precision is often higher.

As mentioned, we compared our results with those obtained using other techniques, which are summarised in Table 7.2. Looking at macro-averaging metrics, we find that they are generally low across all models. Only the Random Forest model demonstrates slightly better performance, with a precision of 0.55. In contrast, the FastLAS model stands out in terms of micro-averaging recall, and it also achieves the highest levels of accuracy and weighted F1 score. Regarding the PSS, during the first period, Random Forest outperforms FastLAS by a considerable margin. In the second period, while the performance gap narrows, Random Forest still maintains the edge.

The line charts graphs presented in Fig.7.1 are comparing accuracy of different models across the 10 folds of cross-validation.

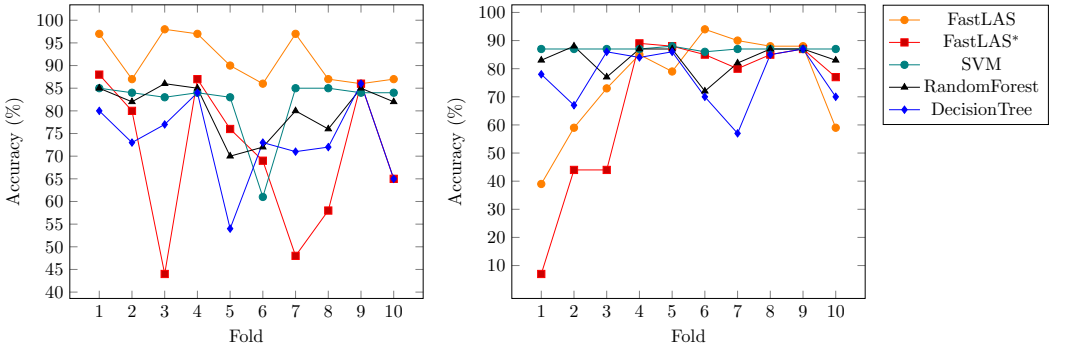


Figure 7.1: Accuracy for each fold for each model. Set_{jfm} (left) and Set_{amj} (right)

For Set_{jfm}, FastLAS consistently maintains a high accuracy throughout most of the folds, with accuracy values generally above 80%. Even if there is some variability, FastLAS remains relatively stable compared to the other models. In contrast, the FastLAS* variant exhibits fluctuations in accuracy. It performs poorly in folds 3 and 7, where accuracy drops to around 50%, but in other folds, its accuracy is comparable to that of the other models. The SVM model, along with Random Forest and Decision Tree models, shows relatively stable performance, with accuracies clustered mostly between 70% and 80%. However, none of these models reach the accuracy levels achieved by FastLAS.

7.2.5 xASP

In this study, we applied xASP to interpret the predictions of rainfall of our system. xASP provides a visual and structured form of explanation by representing the answer set as a directed acyclic graph. This graphical representation allows users to understand how specific conditions lead to a particular rainfall level. Nevertheless, further improvement is needed, as our ultimate goal is to translate the xASP graphs into natural language explanations.

An illustrative example is presented in Figure 7.2, where each node represents a fact (dark green) or derived literal (light green) in the answer set. Edges are directed arrows showing dependencies labelled by the rule that explain this relation. This example, while somewhat complex, is still far from the most intricate xASP graphs our system can generate. In real-world scenarios, the reasoning chains may involve numerous interacting conditions and intermediate deductions, resulting in larger, more tangled graphs. Indeed, the example of Figure 7.2 represents just the justification of the prediction of a timestamp. For this reason, we also report a more complex example with a corresponding zoomed-in view of a predicate (Figure 7.3).

7.2.6 Future Works

This study explored the application of FastLAS (including its extension, FastLAS*) to learn logical rules and then use them to predict and explain rainfall levels. The results demonstrate that FastLAS has considerable potential in weather forecasting, particularly due to its ability to generate explainable and interpretable models, with FastLAS* enabling the generation of rules involving numerical comparisons.

Future work will focus on several directions to further improve the predictive performance and practical applicability of FastLAS. First, we plan to expand the geographic scope of our data collection and incorporate information from a wider range of weather stations and seasons. This will allow us to evaluate the system under more diverse conditions and to consider alternative discretisation levels for rainfall.

Additionally, in situations where multiple stable models are plausible due to incomplete data, we aim to exploit reliable weather prediction systems to identify the most feasible model.

Finally, a key extension will be to translate the xASP graph representations into natural language explanations, producing automatic, human-readable reports that resemble those provided by expert meteorologists in communication media.

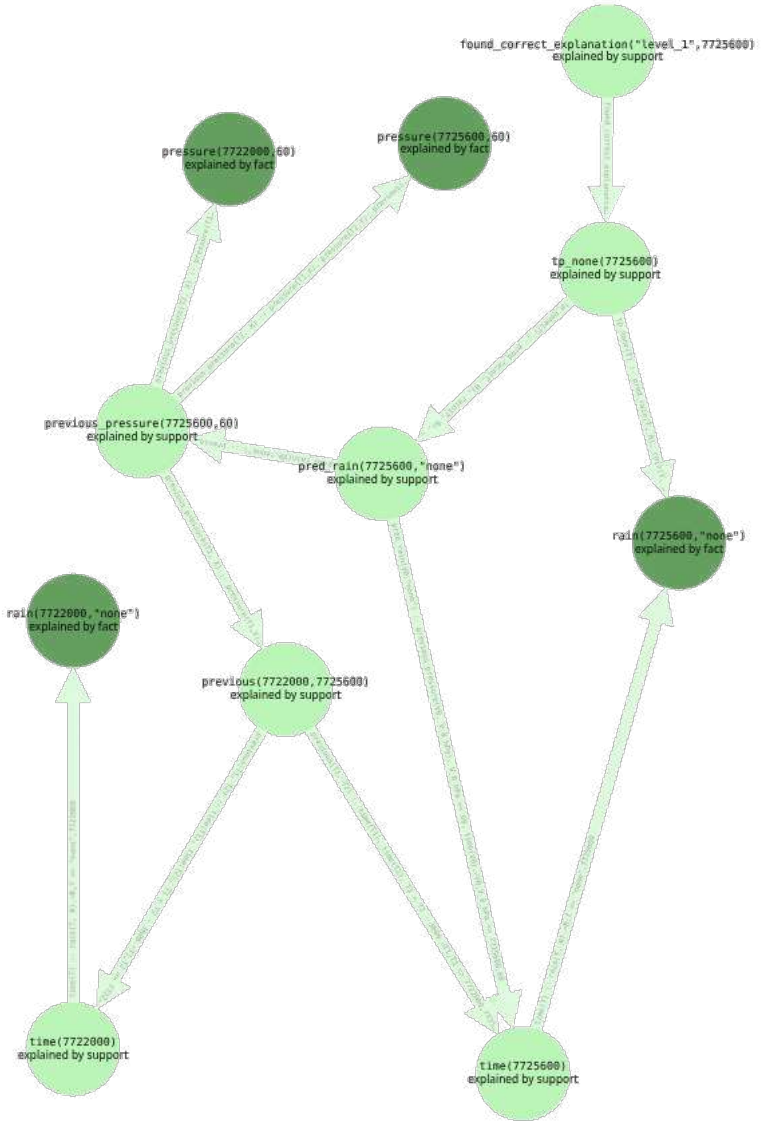


Figure 7.2: xASP graph illustrating why the prediction of level 1 rainfall is correct.

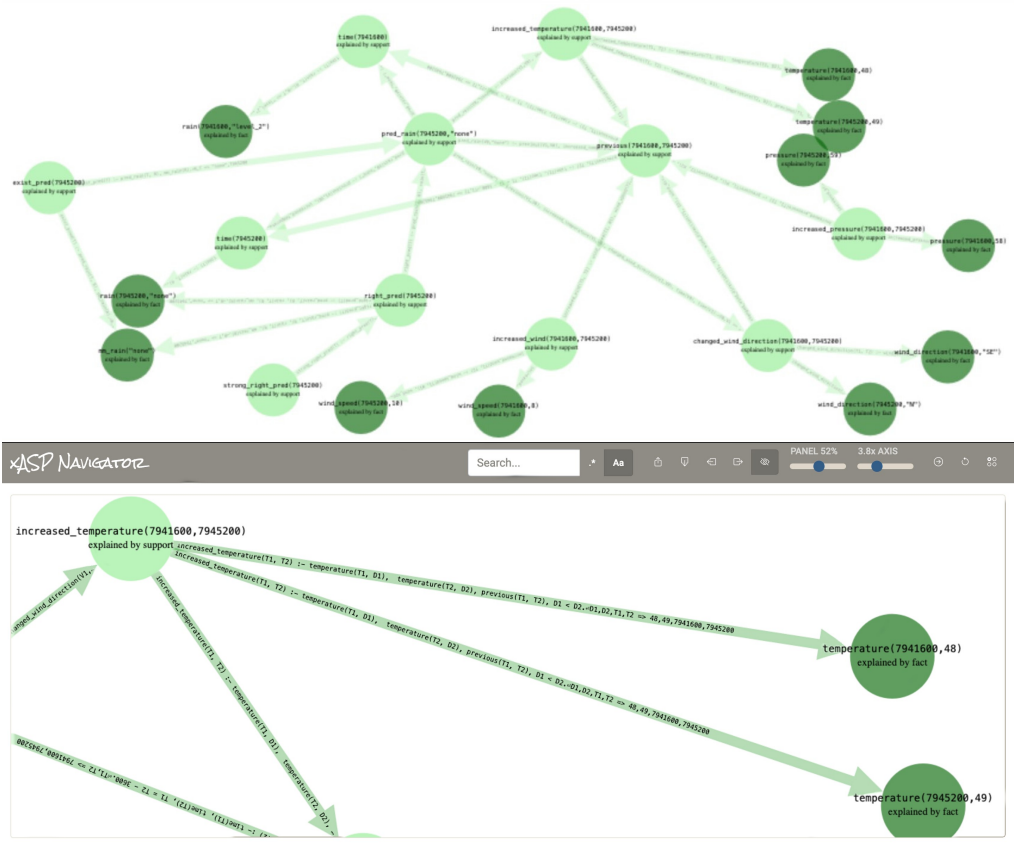


Figure 7.3: Full explanation of the answer set and zoomed-in view of a predicate explanation by xASP. The full graph illustrates a more complex reasoning structure; due to its size, the text is too small to be legible.

7.3 Modelling Advection

In Section 7.2, we noted that data from meteorological stations are insufficient for making long-term predictions, as they only allow forecasting of short-term events, typically within the next hour. In this second study, we propose an alternative approach to long-term weather prediction by employing meteorological variables both at the surface and within the atmosphere.

7.3.1 Dataset

Copernicus European Regional ReAnalysis (CERRA) is a regional reanalysis dataset covering Europe, combining observations and NWP models to create spatially and temporally consistent historical reconstructions of atmospheric and surface variables. The dataset extends from September 1984 to the present and is continuously updated on a monthly basis, thus constituting one of the most complete and detailed reanalysis products currently available for the European domain.

CERRA covers the entire European region, including parts of northern Africa, the Atlantic Ocean, and areas up to the Ural Mountains. The format of data is in GRIB2, a standard widely used for meteorological data. Data in GRIB (*General Regularly-distributed Information in Binary form*) files are organised in the form of a regular grid, where each cell represents a specific geographical location on the Earth's surface, identified by its latitude and longitude. For every cell, the file stores the value of a given meteorological variable corresponding to that location at a given time and, when applicable, at a specified altitude or pressure level. The horizontal resolution is approximately 5.5 km $\times$ 5.5 km for the high-resolution reanalysis, while the ensemble members, used for uncertainty estimation, have a coarser resolution of about 11 km $\times$ 11 km, that is the resolution used for this work. The vertical structure of the dataset is defined on 29 pressure levels, ranging from 1000 hPa (near the surface) up to 1 hPa (upper stratosphere), thereby providing a detailed three-dimensional representation of the atmosphere. The available atmospheric variables on pressure levels include temperature, geopotential, relative humidity, cloud cover, and the zonal and meridional components of the wind.

The dataset includes both reanalysis and forecast fields, depending on the variable considered. Reanalysis is a process that combines observational data with numerical models to produce a complete and physically consistent reconstruction of the state of the atmosphere over an extended period. This integration is achieved through data assimilation, a method also employed in NWP. In data assimilation, a previous forecast is optimally merged with newly available observations to generate an improved estimate of the atmospheric state, known as the analysis. This analysis then serves as the starting point for the next forecast cycle.

The variables considered in this study so far are temperature, wind speed and direction, and cloud cover. Each of these variables is analysed at four different altitudes: 100 m, 3 km, 5.5 km, and 9 km above the surface.

7.3.2 Methods

In meteorology, advection refers to the horizontal transport of an atmospheric property, such as heat or moisture. Mathematically, advection represents the rate of change of a scalar or vector field (for instance, temperature or humidity) resulting from the horizontal velocity of the air mass. It is an important mechanism in atmospheric dynamics and weather forecasting, as it helps explain how air masses interact and how synoptic-scale phenomena such as cold fronts, warm fronts, and cyclones evolve. For this reason, the idea is to model this phenomenon within a logic programming framework, starting directly from the GRIB data.

OSMER experts produce daily weather *bulletins* (see **??**) that explain the forecasted conditions over the region. Their workflow relies on the outputs of multiple numerical weather prediction models based on GRIB data. These outputs are discussed collectively during expert briefings, after which a consolidated forecast and its accompanying explanation are produced. The objective of our work is to automate this briefing phase by introducing an intelligent agent, called SERGIO (**S**imulating **E**xplanation of **R**e**GIO**nal weather forecast), which aims to replicate the explanatory reasoning process typically carried out by human experts.

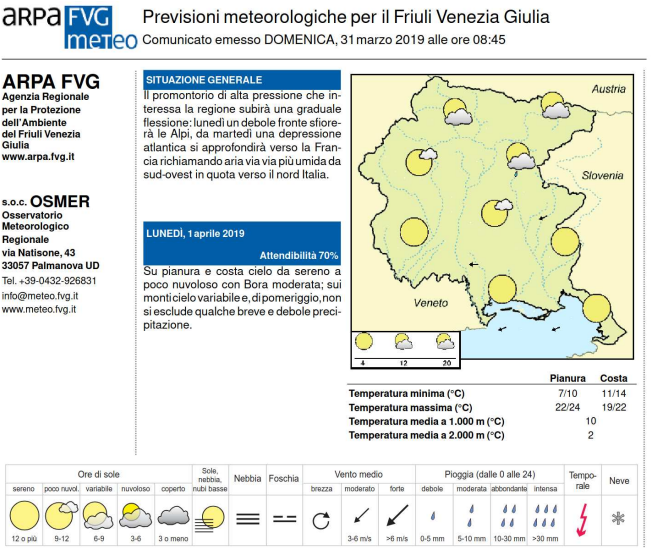
The first step consists in converting the GRIB files into images and restricting them to our region of interest, *i.e.* Friuli Venezia Giulia region. Since each cell in a GRIB file contains the value of a meteorological variable, this value can be interpreted as an intensity level and visualised using an appropriate colour map. An example is shown in Figure 7.6 on the left. The contours of the borders are included solely for visual clarity; however, they are removed prior to the following step.

Once the conversion is completed, a clustering algorithm is applied to the resulting images. The rationale behind this step is that, although the colour map contains many shades representing different intensity values, our primary interest lies in identifying the centres and boundaries of meteorological masses. Therefore, reducing the number of colour levels,typically to four, provides a sufficiently informative representation for our analysis while minimising unnecessary visual complexity. In Figure 7.6, the original visualisation of the GRIB file is shown alongside the corresponding clustered image.

The next step consists in using tobac. Tobac is a Python package designed for the identification, tracking, and analysis of clouds and other meteorological phenomena across various types of gridded datasets. It can track features using any meteorological variable on any grid structure, making it suitable for data from diverse sources such as radar measurements, satellite observations, and numerical weather prediction model outputs. The main steps in the tobac workflow can be summarized as follows:

1. *Feature Detection*: tobac first identifies the specific locations or "cores" of the phenomena being tracked. The user selects a relevant input field (*e.g.*, vertical velocity in a model, or outgoing longwave radiation from a satellite) and defines one or more threshold values. Tobac then detects contiguous regions in the dataset that exceed or fall below these thresholds. Each detected region is represented by a single point location, typically computed as a weighted mean (such as the centre of mass) rather than a simple geometric centre.

2. *Segmentation*: once the feature cores are found, tobac associates each with a distinct spatial area. This step commonly employs watershedding techniques,



General Info
The high pressure promontory affecting the region will undergo a gradual decline: on Monday a weak front will touch the Alps. From Tuesday, an Atlantic depression will emerge towards France, bringing increasingly humid air from the South West, at high altitude towards Northern Italy.
Monday, April 1, 2019
Clear to partly cloudy sky, with moderate Bora winds, on plain and coast. On the mountains Variable skies; some brief and light precipitation cannot be ruled out in the afternoon

Figure 7.4: The bulletin for the 1st of April 2019 for the FVG region issued by OSMER FVG. It includes a pictogram describing the sky coverage and precipitation in different key locations, as well as the overall minimum and maximum temperatures. The report also includes a brief textual description summing up some of the reasons the domain experts considered for the forecast. English translation is reported in the right box.

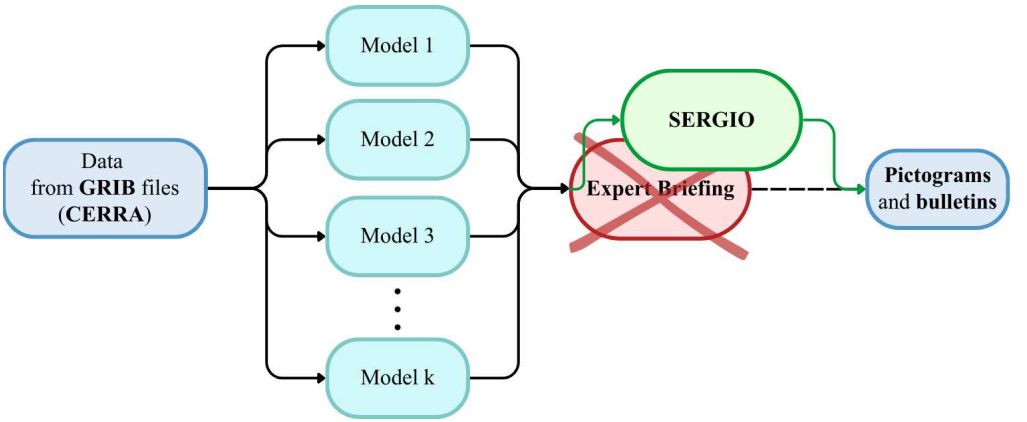


Figure 7.5: Several Deep Learning models are used for (black box) predictions. After the predictions, a group of experts decided how to label the map and writes the sentences to be written in the bulletin. Our tool SERGIO can replace this final stage.

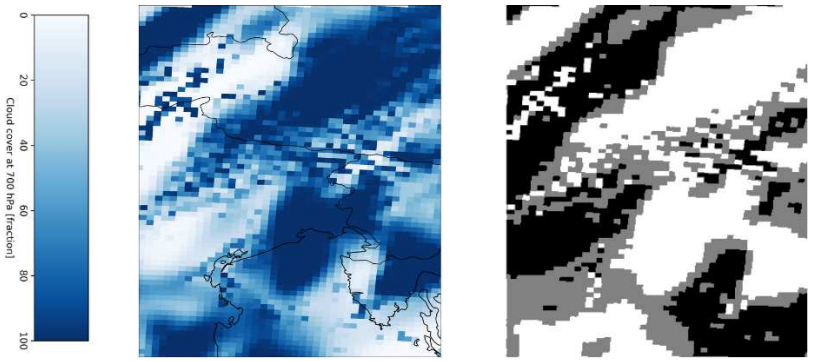


Figure 7.6: Example of the visualization of a GRIB file (left) and the result of the application of clustering on it (right).

which treat the input field as a topographic surface and use the previously detected feature cores as "seed" or marker. The outcome is a mask, that is an array that labels all grid points belonging to a specific feature at a given time step.

3. *Tracking*: tobac then links *features* (the blobs) identified at consecutive time steps into continuous trajectories or tracks. To do so it uses Trackpy, an existing particle tracking library, which predicts the likely position of each feature in the next frame based on its previous motion. Tobac also includes a procedure to manage complex events such as the splitting or merging of tracked objects. This post-processing step employs connectivity analysis (*i.e. Minimum Euclidean Distance Spanning Tree*) to define relationships between parent tracks and their resulting child cells.

4. *Analysis and Output*: the final stage provides tools for analysing and visualising the identified tracks and features.

7.3.3 FastLAS Encoding

The FastLAS encoding is still in progress, so in this section we are going to describe just the context of each example, while the inclusions, exclusions and hypothesis space is left for future works.

From the output of TOBAC, we derive ASP facts encoding the presence, spatial extent, and temporal evolution of detected features at different altitude levels. In particular, facts describe (i) the detection of a feature at a given time and level, (ii) its approximate location, and (iii) its displacement between consecutive time steps. Wind vectors extracted directly from CERRA are used as complementary information and can be related to the observed motion of features, enabling the ILP system to reason about consistency between large-scale flow and observed advection patterns.

We considered the following variables, at six different height levels $1000hPa$, $925hPa$, $850hPa$, $700hPa$, $500hPa$ and $300hPa$ to generate the context of each example:

- *Cloud coverage*: Percentage representing the fraction of a grid cell covered by clouds.

- *Relative humidity*: Ratio, expressed as a percentage, between the actual water vapour pressure and the saturation vapour pressure.
- *Temperature*: Air temperature measured in kelvin.
- *U-component of wind*: East–west component of the horizontal wind velocity.
- *V-component of wind*: North–south component of the horizontal wind velocity.

All variables are extracted from the geographical area bounded by 10 and 15 degrees of longitude and by 44.5 and 48 degrees of latitude. This spatial extent covers the entire Friuli Venezia Giulia (FVG) region while also including a surrounding contextual area. To derive logical facts from the data, heatmaps are generated for all variables (except wind) across all considered altitude levels. For each hour and each level, cloud coverage, temperature, and humidity are represented as frames capturing snapshots of the corresponding simulations. Within these maps, we identify as features spatial regions with blob-like structures that exhibit a significant colour contrast with respect to the background. These features allow us to highlight areas characterised by cloud presence, high humidity, or elevated temperatures. Those facts, as shown in Figure 7.7, constitute the context of a day.

To learn and evaluate the hypothesis, we constructed a dataset consisting of 70 days of data. The selected days span the year 2025, cover all four seasons, and include periods identified by domain experts as exhibiting characteristic seasonal phenomena (e.g. *Sirocco winds* or *spring cold fronts*). This selection, made in collaboration with experts from OSMER, was intended to support the learning of hypotheses that explicitly account for such seasonal effects. All folds were executed on a system equipped with an Intel Core i7-13700KF processor with a base clock of 3.4 GHz (up to 5.4 GHz in boost mode), 128 GB of DDR4 RAM, and an NVIDIA RTX 4090 GPU. The complete data preprocessing pipeline required to generate the LAS examples for the 70-day dataset took approximately four and a half hours. As illustrated in Figure 7.8, the preprocessing phase can scale across multiple CPU threads for data extraction and can offload the heatmap clustering step to the GPU when available.

By explicitly modelling advection within the context of each example, the learned hypotheses are able to capture dynamic phenomena, such as the transport of cloud masses or moist air from neighbouring regions. This approach closely reflects the reasoning typically employed by human meteorologists when justifying forecast bulletins, and represents a key factor in producing explanations that are both faithful to the underlying data and consistent with expert practice.

7.3.4 Results

Currently, we are able to segment (see Figure 7.9) and track (see Figure 7.10) advection of weather variables. In particular, Figure 7.10 illustrates the tracking process over three consecutive timeframes. The image shows the movement of three identified features (numbered 1, 2, and 3, with a partially visible 4) over time. In the first timeframe (0), the initial feature center is indicated by a white triangle to denote that it is the first time it is detected. If a new feature appears at, for example, timeframe 2, then also its centre will be marked with a white triangle. In subsequent timeframes, the tracked centre is

```
cloud_at_100m_covers(pontebba_tarvisio,532, 1).
cloud_at_100m_covers(pontebba_tarvisio,532, 2).
cloud_at_750m_covers(pontebba_tarvisio,517, 1).
...
cloud_at_1_4km_covers(pontebba_tarvisio,680, 0).
cloud_at_1_4km_covers(pontebba_tarvisio,680, 1).
cloud_at_1_4km_covers(pontebba_tarvisio,680, 2).
...
cloud_at_3km_covers(pontebba_tarvisio,777, 5).
cloud_at_3km_covers(gemona_stolvizza,777, 6).
...
cloud_at_9km_covers(trieste,856, 2).

%summing up temperature and humidity facts (one per locaiton)

temperature_at_morning(sappada_forni_villa,278.00).
temperature_at_afternoon(sappada_forni_villa,278.69).
temperature_at_morning(pontebba_tarvisio,278.73).
temperature_at_afternoon(pontebba_tarvisio,277.38).
...
temperature_at_morning(pordenone,277.50).
temperature_at_afternoon(pordenone,276.33).
humidity_at_morning(sappada_forni_villa,34.67).
humidity_at_afternoon(sappada_forni_villa,27.50).
...
humidity_at_morning(pordenone,47.33).
humidity_at_afternoon(pordenone,65.83).
wind_blowing_morning(sappada_forni_villa,SW,4.149).
wind_blowing_afternoon(sappada_forni_villa,S,3.680).
...
wind_blowing_afternoon(pordenone,S,3.417).
```

Figure 7.7: An example of the context of a day.

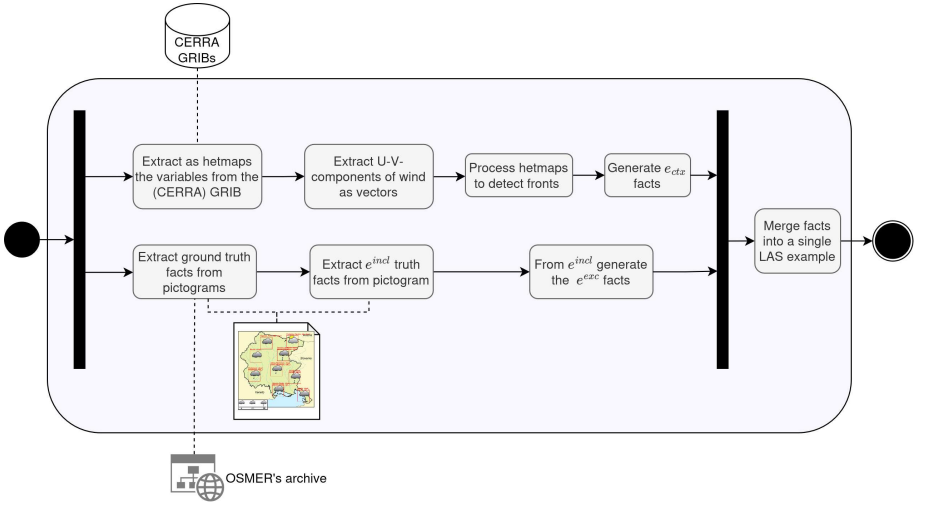


Figure 7.8: An activity diagram describing the process used to generate each WCDPI each example. Note that the context facts in the WCDPI and e^{inc} (as well as e^{exc}) can be computed independently since they are derived from different data sources.

marked by a red square, meaning that it is the centre of a feature already present in the previous timeframe. The blue line extending from the red square shows the trajectory or movement vector to the next time step. The orange dashed circle around each feature represents the search area, *i.e.* the region where the feature's centre is most likely to be found at the next timestamp.

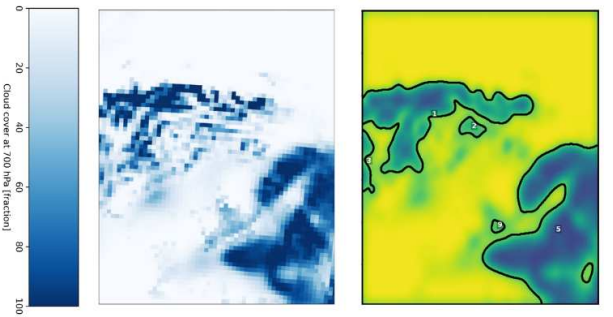


Figure 7.9: Cleaned GRIB image (left) and its segmentation (right).

At the same time, we are conducting experiments with domain experts to model the evolution of advections, with the goal of providing natural language explanations of these phenomena in the same style provided by meteorologists.

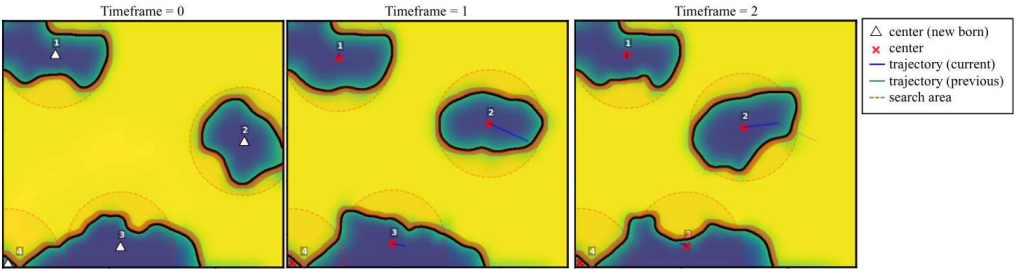


Figure 7.10: Three consecutive timeframe with detected trajectories.

7.4 Lightning Prediction

The third problem we tackled is the one of predicting lightning. Specifically, the deep learning techniques employed for this aim at predicting the base 10 logarithm of the number of lightning strikes occurring within a 6-hour period in the Friuli Venezia Giulia region. This work is a continuation of the work of Soldà [101] and Foschiani [102] in their bachelor degree thesis.

7.4.1 Dataset

Data pertaining to lightning strikes are sourced from CESI-SIRF of Milan, provided to OSMER - ARPA FVG. CESI-SIRF has operated a lightning detection network, known as the *Sistema Italiano Rilevamento Fulmini* (SIRF), since 1994. Initially, SIRF primarily monitored lightning discharges between clouds and the ground. Since 2016, it has also been able to detect intra-cloud lightning strikes. Initially, only the months from April to September were considered due to the higher frequency of storms during this period. Since, data from other months were also incorporated into the analysis, despite the lower climatological occurrence of storms.

Features

The predictors used in this study correspond to the fields provided by the deterministic model *European Centre for Medium-Range Weather Forecasts* (ECMWF), saved at OSMER-ARPA FVG in GRIB 1 format. Those GRIB data are a two-dimensional record composed of messages, each containing a header and data. The header includes metadata such as the name of the meteorological variable, isobaric level, forecast time, date, and statistics (maximum, minimum, mean). The data, representing the variable's values, are organised in a 2D matrix, with each cell referred to as a *gridpoint*. The extracted temporal cadences were as follows: 00:00, 06:00, 12:00, and 18:00 UTC from January 2018 to December 2023. The extracted fields are partly defined on different isobaric levels (particularly the levels of 500, 700, 850, and 925 hPa) and partly defined at the surface or on predefined levels, such as total precipitation or wind estimated at 10 m above ground level. ECMWF provides output from both the deterministic model, called *Integrated Forecasting System* (IFS), and the only one used in our work, that of the probabilistic model, called *Ensemble Prediction System* (EPS). The meteorological

variables were divided into two categories: data on domain called *mesoscale* (48 predictors) and those on the domain called *synoptic* (46 predictors). The synoptic data cover a larger area (from -5° to 25° longitude and from 35° to 55° latitude) and have a resolution of 0.25 degrees, both for latitude and longitude. The mesoscale domain covers a smaller spatial extent compared to the synoptic scale (from 4° to 18° in longitude and from 42° to 50° in latitude), but it is represented at a higher resolution, namely 0.125°. At the mesoscale, extracted variables are defined on a single vertical level that are not available at the synoptic scale, while the vertical velocity variable w, which is provided at four isobaric levels in the synoptic data, is not available in the mesoscale dataset.

In addition to those, derived meteorological variables were calculated from predictors directly extracted from the IFS model. These derived variables were computed for both domains, synoptic and mesoscale, using different horizontal resolutions.

Specifically, for lightnings prediction, the variables of interest are:

- synoptic data, directly extracted from ECMWF for the predictors available at the isobaric levels of 500, 700, 850 and 925 hPa
  - geopotential height $[m]$
  - relative humidity [%]
  - temperature $[K]$
  - east-west wind $[m/s]$
  - north-south wind $[m/s]$
  - upward wind $[Pa/s]$
- mesoscale data, directly extracted from ECMWF for the predictors available, for the first 5 variables, at the isobaric levels of 500, 700, 850 and 925 hPa, and for the last variable, at just one level (ground variables)
  - geopotential height $[m]$
  - relative humidity [%]
  - temperature $[K]$
  - east-west wind $[m/s]$
  - north-south wind $[m/s]$
  - temperature at 2 meters from surface $[K]$
  - amount of convective precipitation $[m]$
  - amount of total precipitation $[m]$
  - east-west wind at 10 meters from surface $[m/s]$
  - north-south wind at 10 meters from surface $[m/s]$
  - mean pressure at sea level $[Pa]$
- variables derived from predictors extracted directly from IFS model
  - mixing ratio $[g/kg]$
  - equivalent potential temperature (Θ_e) $[K]$

- equivalent saturated potential temperature $[K]$
- zonal water vapor flux $[kg/(sm^2)]$
- meridional water vapor flux $[kg/(sm^2)]$
- maximum buoyancy $[K]$
- lapse rate of temperature $[K/km]$
- lapse rate of Θ_e $[K/km]$
- downdraft Potential $[K]$

In order to make the domains of the predictors more comparable, also reducing the influence of the original variable ranges on the gradient descent function, the data were normalized before getting fed to the network. The normalization was made using standardization, *i.e.*applying the following formula for each grid-point:

$$x_{i,j} = \frac{\hat{x}_{i,j} - \mu}{\sigma} \tag{7.2}$$

where μ and σ represent mean and standard deviation.

7.4.2 Architecture

The weather prediction, as we mentioned, is a complex phenomena that involves many variables. Nowadays, there are some mathematical models that are able to handle most of them and make predictions. Given this, the use of advanced computational tools and modelling techniques has become indispensable.

The proposed architecture adopts a hybrid design that integrates convolutional neural networks (Section 5.4.1) with recurrent neural networks (Section 5.4.2), specifically long short-term memory units, in order to capture both spatial and temporal patterns in the data. In the context of weather forecasting, CNNs can be applied to analyse satellite imagery, radar data, and other spatially distributed observations. By extracting meaningful patterns and spatial dependencies from these data sources, CNNs can improve the accuracy of weather prediction models, especially for short-term forecasts and localized phenomena such as thunderstorms and convective systems. Instead, RNNs are designed to model sequential data and capture temporal dependencies over time. This means that they can be used to make predictions about future data points based on past data points. Unlike traditional feedforward neural networks, RNNs have recurrent connections that allow them to maintain internal memory and process sequences of arbitrary length. This memory allows them to store information about past inputs, which can then be used to make more accurate predictions about future inputs. This makes them useful for time series forecasting tasks, including weather prediction. In weather forecasting applications, RNNs can be trained on historical weather data to learn complex temporal patterns and relationships between atmospheric variables. By exploiting this learned knowledge, RNNs can make accurate predictions of future weather conditions.

The first stage of the network consists of three convolutional blocks, each comprising a two-dimensional convolutional layer followed by a *Scaled Exponential Linear Unit* (SELU) activation and a max-pooling operation. The convolutional layers progressively

increase the number of filters while reducing the spatial resolution. Following convolutional processing, the extracted spatial features are reshaped into a sequence of fixed length. This transformation enables the network to treat spatial feature maps as temporally ordered inputs to the recurrent component. An LSTM layer then processes the sequence, modelling dependencies across time steps and mitigating issues of vanishing or exploding gradients through its gating mechanisms. The final stage of the architecture consists of a fully connected output layer. The LSTM's hidden representation at the last time step is projected through a linear transformation and activated with SELU, producing a one-dimensional output for each sample in the batch. This design makes the architecture suitable for regression or binary classification tasks, where the aim is to predict a single value from spatio-temporal input data.

It was also implemented a variant of this network which uses ConvLSTM (Section 5.4.3) instead of simple LSTM. Convolutional LSTM networks were introduced by a research group led by X. Shi [103] who proposed a novel machine learning approach for precipitation forecasting in a region of Hong Kong. The use of fully connected LSTM layers for spatially gridded data presents an important limitation: it requires an extremely large number of connections, as separate weights must be learned for each grid point and its corresponding state. To address this issue and reduce the number of parameters, convolutional operations were incorporated directly into the LSTM architecture.

7.4.3 Future Works and Conclusions

The best results reported in [101] were $R = 0.68$ on the validation set and $R = 0.71$ on the test set for the synoptic scale, and $R = 0.74$ for both validation and test sets at the mesoscale, where R is the *Pearson correlation coefficient*[1]. They also found out that the most relevant predictors for the synoptic data were Θ_{ES} at 925 hPa, $MB_{925-500}$ hPa, wa at 700 hPa, and Θ_{ES} at 850 hPa, while for the mesoscale they were c_p, $MB_{925-500}$, and Θ_{ES} at 925 hPa.

In our experiments, the results obtained with a convolutional recurrent neural network were comparable to those previously reported, and are therefore not presented in detail here. Nonetheless, the approach shows considerable potential, although further work is required to consolidate and extend these findings. To this end, a substantial refactoring of the codebase has been undertaken to improve modularity and scalability. In parallel, we are evaluating the inclusion of data from additional years to enhance the robustness of the model. Furthermore, we plan to investigate the use of encoder–decoder transformer architectures as a possible alternative to the current framework.

Once satisfactory predictive performance is achieved, we intend to apply ILP, as in our previous works, to learn symbolic rules describing the identified events and to employ them for providing interpretable explanations of the model's behaviour.

[1] correlation coefficient that measures linear correlation

8

Juridical Reasoning

While examining the application of ILP in the juridical domain, it is clear that some of the most notable contrasts between formal logic systems and human cognitive processes originate in contexts involving perception, ambiguity and common sense. Frequently, the common issues between all these contexts are the reliance on incomplete or uncertain knowledge and vague concepts. In legal reasoning, for example, perceptual biases may lead to missing important details or common sense approaches may omit rare exceptions.

These complexities deal with the notion of *open texture*, a concept firstly introduced by Hart [104] in 1961, and then revisited later in 2012. His idea highlights the inherent indeterminacy of legal provisions, arguing that no matter how precise or exhaustive a law may appear, a degree of vagueness will always persist, allowing for reinterpretation and contestation. Specifically, he states that:

> *The open texture of law means that there are, indeed, areas of conduct where much must be left to be developed by courts or officials striking a balance, in the light of circumstances, between competing interests which vary in weight from case to case.*
>
> (H. L. A. Hart)

This perspective aligns with the challenges faced by computational models attempting to formalise legal judgment, as they must account for the context-dependent nature of legal reasoning rather than relying only on rigid logical structures.

In this chapter, we will propose an approach that addresses the challenges posed by vague concepts and we will also explore the methods employed to extract rules from historical judgments. Moreover, we provide a potential modelling of certain articles from Italian law to support and validate the methodology applied in this thesis.

All the code used in this work is available from `http://clp.dimi.uniud.it/sw/`, while our contributions are formalized in [105, 106, 107, 108].

8.1 Related Works

The intersection of logic programming and legal reasoning has a rich history, beginning with foundational work in the mid-20th century. Allen [109] pioneered the interpretation of legal documents through symbolic logic in the 1950s. Decades later, Allen and Saxon [110] detailed two critical features of legal rules that require careful attention in logical encodings. The first one is the ambiguity of natural language in legal texts. The work of Kowalski et. al [111] provided a bridge between human-readable language and formal logic to address this issue. This approach complements reasoning systems by making legal rules more accessible while ensuring accurate logical representation. The second issue is that rules can sometimes be applied even when their preconditions are not fully proven. Additionally, certain rules may allow for commonsense exceptions. While this posed important challenges for expert systems in the 1980s, it is now a natural feature of reasoning systems based on stable model semantics.

Some decades after the work of Allen and Saxon, in the 1980s, Kowalski and Sergot [112] classified legal rules into definitional, normative, and case law. They demonstrated that logic programming could represent significant portions of British law, including the 1981 British Nationality Act, using Horn clauses and their extensions. However, their reasoning modules struggled with unstratified uses of default negation. Notably, legal systems differ considerably across countries. In the UK, law primarily relies on precedents, encouraging the application of machine learning to identify patterns. However, these methods often fail to capture the underlying logic behind decisions. In contrast, countries like Italy base their legal systems on formal codes, using precedents only to address ambiguities. This distinction highlights the potential for machine learning to identify exceptions rather than general principles.

Golshani [113] later argued that automated legal reasoning systems should not aim to provide definitive answers. Instead, their role is to construct well-reasoned arguments, leaving the final decision to judges while enabling the analysis of hypothetical scenarios.

All this considered, a logic-based approach, such as Inductive Logic Programming, offers distinct advantages for legal reasoning. Modern ASP solvers, like those based on *Conflict-Driven Clause Learning* (CDCL), enable efficient reasoning while tools like ILASP simplifies the learning of rules and exceptions in non-monotonic domains. Recent work by Sartor et al. [114] demonstrates how Logical English, used as an intermediate language, aids in encoding laws into ASP programs. Their approach employs the *solver for Constraints Answer Set Programs* (s(CASP)) solver, which provides top-down reasoning and explanations by tracing solution paths rather than constructing complete models. However, top-down ASP solvers face challenges in scalability and inconsistency handling. They lack the pruning capabilities of CDCL techniques and require additional program analysis to detect inconsistencies in unstable models, limiting their practical applications in complex scenarios.

8.2 Problem

Legal reasoning often involves interpreting complex rules, handling exceptions, and considering precedents, making it a challenging task. Traditional methods rely on human expertise, which can lead to inconsistencies and subjective interpretations. By encoding

legal rules with ASP, we aim to formalise legal reasoning, ensuring a structured and transparent decision-making process, even if we still leave the final verdict to judges. Indeed, automating legal reasoning can serve as a valuable tool for lawyers, assisting them in assessing different legal arguments and improving decision-making efficiency.

To enable automated legal reasoning, laws must first be encoded in a formal representation. One key aspect is that certain legal concepts remain consistent across different laws. For example, principles such as "theft" or "violence" may appear in multiple jurisdictions with slight variations. By encoding these shared concepts in ASP, we aim to create an homogeneous foundation. Another advantage of ASP-based legal reasoning is the ability to simulate multiple legal scenarios. Given a specific set of facts, different legal interpretations or judicial decisions may lead to varying outcomes. Using ASP enables this exploration of different possible consequences, helping legal operators to evaluate the implications of their decisions.

Additionally, legal reasoning is often influenced by precedents and past judgments, particularly in case law-based systems. Encoding previous court decisions allows the system to learn patterns and identify legal arguments that led to a certain case in the past and that may be applied also to new, similar, cases. The solution we propose is a system that can generate plausible decision scenarios, each accompanied by an explanation. These scenarios can assist a judge in making an informed decision or provide a logical justification for a decision made autonomously.

8.2.1 Vagueness

In legal argumentation, a significant challenge emerges when deriving the decision for a specific case from a general and abstract legal rule. This sometimes requires distinguishing between two distinct phenomena: *uncertainty* and *vagueness*. Uncertainty has an epistemological issue, that is the lack of information regarding a particular event or circumstance. For example, if we say "Rebecca might be at home", we are expressing uncertainty because we do not have enough information to determine Rebecca's exact location at the moment. In contrast, vagueness occurs when sufficient information is available, yet indeterminacy arises due to semantic ambiguity. To better illustrate this, consider a classical example of vagueness: while it might be evident that a man with zero hair is bald, no fixed number of hair can be used as a threshold for deciding baldness vs non-baldness. There are borderline cases of baldness, making it indeterminate whether the statement "x is bald" is true or false.

To address interpretive challenges, legal science has developed a variety of strategies, which differ depending on the legal system in question. These strategies include recourse to legal principles (*analogia iuris*), references to other norms within the same legal field (*analogia legis*), or reliance on judicial precedents, which are binding in certain systems, such as *common law*. Although these tools provide solutions, they still allow for interpreter discretion, which can reduce confidence in the impartiality of justice.

8.3 Dataset

This research focuses on the systematic codification of articles from the *Italian Penal Code* (abbreviated in `cod.pen.`). The primary scope, within the `cod.pen.`is in

the *Titolo XII del Libro Secondo*, which regards "crimes against the person". The analysis includes many articles, but the main focus is on: theft (`Art. 624`), snatch theft (`Art. 624-bis`), robbery (`Art. 628`), beatings (`Art. 581`) and personal injuries (`Art. 582`). The study also examines `Article 56`, which addresses attempted crimes, given its frequent reference in legal judgments. To ensure consistency and reliability, relevant judicial rulings associated with these articles have been identified and analysed, also with the help of legal experts. This process aims to evaluate the logical and legal coherence of the outcomes in relation to established precedents.

Those articles constitute the background knowledge of our ILP system, while as examples, we incorporated a set of decisions from the *Court of Cassation*[1], which plays a central role in unifying legal interpretation within the Italian judicial system. This step involved a manual encoding process that we aim to automatize in the future. The model was then tested using an additional set of Court of Cassation decisions.

We will now report examples of the encoding process that we used to define articles in ASP. In `cod.pen.`, the concept of "*furto*" (theft) can be translated as follows:

Article 8.3.1 (`Art.624`)**.** *Whoever takes possession of another person's movable property, capturing it from its owner, in order to gain profit for himself or others [...] is guilty of theft.*

The key element for being accused as *reo*[2] of such crime are: a movable property, an action of abduction from the victim, an action of taking possession by the agent, and the intention of profit.

Then we have two specialization of this article, which are "*furto con strappo*" (theft by snatch, `Art. 624-bis`) and "*rapina*" (robbery, `Art. 628`).

Article 8.3.2 (`Art.624-bis`)**.** *Whoever takes possession of another person's movable property, capturing it from the person holding it, in order to gain profit for himself or others, snatching it out of the person's hand or from the person's body [...] is guilty of theft by snatch.*

Article 8.3.3 (`Art.628`)**.** *Whoever, in order to procure for himself or others an unjust profit, by means of violence to the person or threat, takes possession of another person's movable property, capturing it from the person who has it [...] is guilty of robbery.*

Indeed, both cases require the foundational elements of theft but they also include some additional conditions. Specifically, the presence of violence during the act determines the application of `Art.628` instead of `Art.624`. However, the notion of violence itself is inherently vague, making it challenging to determine whether it applies to a specific situation. Furthermore, in many cases the discrimination between violence or not was represented by the item's "*adherence*" to the victim's body [115, 116]: if considered particularly adherent, then the act of violence transitions from being directed solely at the item to involving violence against the individual, since the thief has to overcome its resistance.

[1] in italian "Corte di Cassazione"
[2] from Latin reus that is *accused*, *guilty*

8.4 Methods

The approach consists of encoding the articles of `cod.pen.`in ASP to construct a program, P_{law}, that formalises juridical principles and legal knowledge. Once P_{law} is defined, it undergoes iterative improvement to enhance its accuracy and consistency. A set of known criminal cases, retrieved from the databases Foroplus[3] and DeJure[4], is manually analysed and encoded as sets of facts, denoted as $C_1, \ldots, C_k$. Each case C_i $(i = 1, \ldots, k)$ is composed by a set (conjunction) of atoms and, for each of them, the program $P_{law} \cup C_i$ is executed to compute its stable models. In particular, consider $C = \{A_1, \ldots, A_k\}$. What we should do is, for $i = 1, \ldots, k$, run the ASP solver on $P_{law} \cup \{A_i\}$:

- If no stable model is found, then it means that there is some inconsistency indicating that some rules in P_{law} may be overly restrictive and require modification. In this case, the issue is examined, and the program is modified accordingly by adding rules relating to different atoms.

- Otherwise, if a stable model is obtained, a further validation step is performed. This step consists in introducing a denial for each atom A_j of C_i, *i.e.* in the form `:- not` A_j. The new program is then run to check whether the expected model is obtained. If the expected outcome is not obtained, this suggests that some rules in P_{law} may be too permissive and require further modification.

Once the model produces the correct classifications, an additional analysis is conducted to determine whether alternative models emerge when assuming only a subset of the facts in C_i. This process may reveal exceptions that necessitate extending P_{law} with additional rules.

Given that legal interpretations and precedents evolve over time, maintaining P_{law} as an up-to-date legal model requires an adaptive refinement mechanism, ideally automated. ILASP provides a framework for learning new rules from judicial rulings, allowing P_{law} to dynamically expand. However, human intervention may still be required to resolve inconsistencies that arise from conflicting legal updates.

8.4.1 Background Knowledge

The initial step in modelling a problem with ILASP involves defining the background knowledge, which, in this case, consisted of translating legal articles into ASP language. This will be the initial P_{law} program. For example, Article 8.3 was represented with the following ASP code:

```
% res: movable property
res(C) :- physical_object(C).
res(C) :- energy(C), economic_value(C).
```

[3] `https://www.foroplus.it/home`
[4] `https://dejure.it/#/home`

```
% a theft happens when something is subtracted from an agent to another
theft(R, V, C) :- subtract(R, C), own(V, C), theft_intention(R),
                  take_possession(R, C), agent(V), agent(R), res(C), V!=R.
```

Similarly, the encoding was applied to the other articles in a consistent manner, translating their content into ASP rules to form the complete background knowledge.

Of particular interest is the case of vagueness found within the articles. To address this vagueness, we chose ASP over alternative logic-based formalisms like many-valued or fuzzy logics: while ASP's ease of modelling is an advantage, there are additional reasons for this decision. Many-valued logics could represent varying degrees of "closeness" to vague concepts like robbery and theft, but this introduces gradations of truth into both reasoning and outcomes, which is unsuitable for the legal context. Instead, we aim for solutions that produce unequivocally true outcomes while allowing for multiple alternative solutions, where applicable.

We modelled vague concepts by assigning a level that quantifies the degree of association with one of two extreme interpretations of the concept. This approach allows for a nuanced representation while maintaining clarity in distinguishing between different degrees. Let us revisit the distinction between theft by snatching and robbery, which lays on the exercise of violence towards another person. Note that, violence here is defined as any action resulting in harm between two distinct agents, excluding self-inflicted harm. Determining whether an act qualifies as violence is not always straightforward. For example, in the case decided on 11 May 2023 by Section 1 of the Bologna Court ("Tribunale"), the stolen object was a wristwatch. The court determined that the watch's tight adherence to the victim's wrist was sufficient to imply that the thief ignored the likelihood of causing harm during the act. This close connection between the object and the victim was considered enough to classify the act as robbery rather than theft by snatching. The case has been encoded in ASP as represented below.

```
violence(R, V) :- damage(R, V), agent(R), agent(V), V!=R.
snatch(R, C) :- loose_physical_adherence(V,C), subtract(R,C), agent(V).
subtract(R, C) :- snatch(R, C), agent(R), res(C).
```

The important aspect of this reasoning is that violence is absent as long as the perpetrator does not foresee it. This foresight has been modelled using the predicate `foreshadowing_violence(R)`, which, when true, indicates that `R` is aware their actions might result in violence. Since an act can result in violence without being foreseen, we introduced a distinction between voluntary and involuntary violence. Additionally, in `cod.pen.`, violence may be directed either towards a person or exclusively towards the object being stolen, providing a more varied representation of potential outcomes.

- towards people, denoted by atoms of the form `person_violence(R,V)` or of the form `indirect_violence_person(R,V)` if violence is only foreseen, and
- towards objects, modelled by atoms of the form `res_violence(R,C)`.

The Court of Cassation in sentence n. 49832 delivered on Dec. 11, 2013 by the Section 2, Criminal Law, has clarified that when an object is attached to the victim's body, the distinction between robbery and snatch theft lies in the degree of adherence. Slight

adherence suggests snatch theft, whereas major adherence indicates an intent to harm the physical integrity of the property, thus constituting robbery. This interpretation provides the basis for the following rule:

```
person_violence(R, V) :- tight_physical_adherence(V, C), subtract(R, C),
                         agent(V), R!=V.
```

The same legal reasoning also establishes that an act initially constituting snatch theft can evolve into an attempted or completed robbery if the perpetrator encounters resistance from the victim. This transformation typically occurs when the perpetrator encounters resistance from the victim, necessitating the application of force or violence. This situation leads to various legal classifications:

- if the perpetrator halts their actions immediately in response to the victim's resistance, the offence is classified as *attempted snatch theft*. This scenario is represented in the model with the following rule:

  ```
  attempted_theft_snatch(R, V) :- resistance(V, R), interruption(R).
  ```

- if the perpetrator attempts to overcome the victim's resistance but ultimately fails and abandons the act, the offence is classified as *attempted robbery*. This scenario is represented in the model with the following rule:

  ```
  attempted_robbery(R, V) :- resistance(V, R), res(C), not interruption(R),
                             not take_possession(R, C).
  ```

- if the perpetrator successfully overcomes the victim's resistance and completes the act, it constitutes a *robbery*. This situation is captured in the model with the following rule:

  ```
  robbery(R, V) :- theft(R, V, C), person_violence(R, V).
  robbery(R, V) :- theft(R, V, C), threat(R, V).
  ```

What has been observed so far does not seem to pose issues of vagueness, except in the determination of whether violence has actually occurred. Specifically, the definitions of tight/loose adherence appear to be highly subjective. Therefore, we have decided to further refine this concept by assigning it a level. In particular, we introduced the following rule:

```
1{adherence(S, C, L): level(L)}1 :- unknown_adherence(S, C).
```

This rule uses a syntactic extension of the base ASP language, that is the choice rule. Intuitively, the head of a choice rule is true if exactly one among the possible atoms of the form `adherence(S,C,L)` is true (for each possible choice of values `S`, `C`, as determined by the predicate `unknown_adherence`). Its effect is to model non-deterministic choice

among a set of alternatives. That is, for each pair S, C, there is exactly one value L which makes the atom adherence(S, C, L) true. In other words, given S and C there is only one possible level of violence L. This kind of rule can be implemented via simple rewriting using simple ASP rules involving default negation [117].

A threshold can be used to propagate the non-deterministic choice:

```
tight_physical_adherence(S, C) :- adherence(S, C, L), agent(S), res(C), L>2.
loose_physical_adherence(S, C) :- adherence(S, C, L), agent(S), res(C), L<3.
```

This way, different assumptions are generated and propagated when the level of adherence may not be predetermined (unknown_adherence). Consequently, a stable model will be generated for each potential adherence level.

The modelling described so far does not refer to any specific law or judgment but serves as a support: a judge is still needed to determine the actual level of a particular adherence. The assistance lies in being able to have a kind of classification of adherence in past judgments, so as to place the new case in the most coherent manner possible. In our suggestion, we have decided to consider as tight the adherence with level equal to or greater than 3, and loose those below this value. Clearly, any other reasonable choice can be made.

8.4.2 Examples

The examples were primarily derived from judgements issued by the Court of Cassation, with preference given to decisions from higher judicial authorities, such as the Criminal Division of the Court of Cassation, when similar concepts were addressed. The legal databases *DeJure* and *ForoPlus*, mentioned previously, were extensively utilised, proving essential for identifying case law and rulings relevant to the encoded articles.

Every example consists of a real judgment and so it must be a positive example. In particular, the description of the case, such as who did what, is represented by facts within the context. The perpetrator and the victim can be considered always the same (their names are not relevant), and so are instantiated in the background knowledge with the facts: agent("R") and agent("V"). The inclusions specify the crimes resulting from the scenario, *i.e.* the final judgment, while the exclusions define the predicates that must not appear in the resulting conclusions. Consider the following judgment:

> *The conduct of the co-defendants, consisting of the removal of the handbag from the victim, constitutes the crime of robbery rather than theft by snatching. This is the final act of a joint offence where both individuals used violence to assault the victim and then fled with the loot, carrying out the act without interruption.*
>
> (Court of Appeal of Rome, Section IV, 03/10/2023, No. 9885)

It has been translated into:

```
#pos({
        robbery("R","V","handbag")
    },{
        theft_snatch("R","V")
```

```
    },{
        theft_intention("R").
        own("V","handbag").
        subtract("R","handbag").
        take_possession("R","handbag").
        assault("R","V").
    }).
```

In fact, the inclusions refer to the final judgment, where the defendant is found guilty of robbery. In contrast, the exclusions assert that it was theft by snatching, a claim that is explicitly rejected in the verdict. The context encompasses the entire sequence of events surrounding the theft.

We ran ILASP (version 4) on different sets of encoded legal articles listed below:

- **Set**$_{AC}$: 583
- **Set**$_{CP}$: 575, 579, 584, 588, 589, 589-bis, 59, 595, 609 bis, 610, 614
- **Set**$_{B,PI}$: 581, 582
- **Set**$_{T,R}$: 624, 624-bis, 628

With the exception of the largest set Set$_{CP}$, which had 70 examples, we encoded about 22 judgments as examples for each set. The size of the hypothesis space, denoted by $|S|$, represents the number of rules and is detailed in Table 8.1. Some sets, indicated with *, use mode declarations to define the hypothesis space, while others rely on a predefined space.

8.4.3 Hypothesis Space

The hypothesis space was generated using mode declarations. For the head of the rules we considered not only the possible judgments (`theft_snatch`, `attempted_theft_snatch`, `robbery`, ...) but also some other auxiliary predicates (`violence`, `damage`...) as these concepts were sometimes vague and required further clarification. Regarding the predicates to be included in the body of the rules, we considered nearly all the predicates appearing in the contexts of the examples, as we cannot predict which one will eventually determine one verdict over another.

Unfortunately, this approach is not sustainable as more judgments are added. Each case is highly specific and typically introduces new predicates. Adding these new predicates to the mode declarations, especially in the body, will quickly result in an excessively large hypothesis space that will saturate the memory.

8.4.4 Results

The rules that ILASP was able to learn are numerous. We will list only those pertaining to the articles of theft and robbery.

```
damage(R, V) :- assault(R, V).
damage(R, V) :- part_of_body(V, P); twist(R, V, P).
damage(R, V) :- make_fall(R, V); drag(R, V).
damage(R, V) :- tug(R, V); drag(R, V).
threat(R, V) :- selfdetermination_limitation(R, V).
theft_intention(R) :- seal(V, O); force(R, O).
theft_intention(R) :- force(R, O); open(R, O).
damage(R, V) :- R != V; res(C; agent(V); agent(R); not res_violence(R, C);
                part_of_body(V, P); block(R, V, P).
```

After learning the legal rules, we introduced another predicate, `using_judgments`, to indicate when a decision is influenced by a prior judgment. For example the rule:

```
damage(R, V) :- part_of_body(V, P); twist(R, V, P).
```

became

```
2{damage(R, V), using_judgment}2 :- part_of_body(V, P); twist(R, V, P).
```

This distinction is crucial in the Italian legal system, where past judgments do not become binding law as they do in the UK. Instead, they serve as persuasive guidance rather than legal precedent. By explicitly encoding this aspect, we emphasize the advisory nature of previous rulings while maintaining a clear distinction between judicial interpretations and statutory law. Indeed, the new rule ensures that any answer set that has triggered a rule learned from ILASP, will also contain the predicate `using_judgments`.

As said, we used both hypothesis space definition and mode declarations techniques. In particular, mode declarations enable a wider range of possible rules but come with a higher computational cost. In fact, for sets that do not employ mode declarations, adding them would make learning infeasible due to excessive memory demands. This difficulty arises because legal rules are highly specific and introduce a vast number of predicates, making it particularly challenging to generate an appropriate hypothesis space automatically. In some instances, we adopted a hybrid strategy, combining both approaches. While certain rules were manually encoded, more general ones were generated using mode declarations. This method strikes a balance between the expressiveness of the hypothesis space and computational feasibility.

The results in Table 8.1 illustrate the high computational cost of using mode declarations. For example, in $\text{Set}^*_{B,PI}$, generating the hypothesis space and performing conflict analysis take considerably longer (204.18 s and 291.99 s, respectively) compared to predefined rule-based configurations. In contrast, predefined sets like $\text{Set}_{B,PI}$ and $\text{Set}_{T,R}$ run much faster at all stages, offering greater efficiency but with less rule flexibility.

All tests were performed on a system equipped with an Intel 12th Gen Core i7-12700 processor (RAM of 16 GB, 4.9 GHz max clock speed), an NVIDIA GeForce RTX 3060 GPU, and an SK Hynix NVMe SSD. The operating system is Ubuntu (24.04.2 LTS) and clingo version is 5.6.2.

Table 8.1: Timing breakdown (in seconds) for different ILASP stages across datasets. The shaded row indicates instead the dimension of the hypothesis space (number of rules).

	$\mathbf{Set}^*_{AC}$	$\mathbf{Set}_{CP}$	$\mathbf{Set}_{B,PI}$	$\mathbf{Set}^*_{B,PI}$	$\mathbf{Set}_{T,R}$
$\|S\|$	5418	531	21	93996	11
Pre-processing	0.016	0.028	0.023	0.046	0.014
Hypothesis space generation	0.294	0.102	0.009	204.179	0.005
Conflict analysis	4.289	1.876	0.087	291.992	0.076
Counterexample search	0.048	0.676	0.099	0.179	0.088
Hypothesis search	1.372	0.311	0.022	52.092	0.010
Total	**6.075**	**3.005**	**0.240**	**549.576**	**0.196**

Contradictions' Handling. During the manual encoding process, contradictions between legal rulings were identified. To resolve these discrepancies, an initial strategy involved prioritising them using the "*Massime di Interpretazione*", a set of interpretative principles designed to handle conflicting legal arguments [118]. These principles fall into three main categories:

- "*Lex specialis*": specific rulings take precedence over more general ones.
- "*Lex superior*": norms issued by higher authorities are given priority.
- "*Lex posterior*": recent verdicts have priority over earlier ones.

However, in some cases, these principles themselves led to conflicting outcomes, making it impossible to determine a clear precedence. To explicitly represent such inconsistencies, the predicate `contradiction` was introduced. This predicate is triggered when two opposing rulings emerge, *i.e.* where one set of facts supports a particular offence, while an alternative judgment suggests a different classification.

A notable example is the classification of "*cervicalgia*" (neck pain). A ruling from the Tribunale Bari sez. I (26/08/2022, n. 3684) states that "cervicalgia" is merely a symptom of pain and not a medical condition. In contrast, an earlier decision from a higher authority (Cassazione penale sez. II, 12/03/2008, n° 15420) classifies "cervicalgia" as an illness. This contradiction arises because Lex posterior would favour the more recent ruling, while Lex superior gives precedence to the higher court's decision. Under the first ruling, if an offender caused "cervicalgia", the crime would not qualify as personal injuries. However, under the second ruling, it would, since "cervicalgia" is recognised as an illness. This inconsistency is represented in the system as follows:

```
contradiction("not illness", "illness", I) :- only_pain(I), illness(I).
```

Another case of contradiction emerged when court rulings conflicted with the legal code itself. One example involves cases of personal injuries occurring alongside snatch theft. According to Articles 624-bis and 628, the distinction between snatch theft and robbery depends on whether violence or threats are directed at the victim. Since personal injuries inherently involve violence (Tribunale Nocera Inferiore, 23/06/2020, n.

551), they should logically be associated with robbery rather than snatch theft. Despite this, some judicial decisions categorised such cases as snatch theft rather than robbery. In our encoding, the legal facts activated the rule for `robbery`, yet the actual verdict classified the offence as snatch theft. When both predicates are present, a contradiction arises, activating the `contradiction` predicate to point out the inconsistency between the legal framework and judicial interpretation.

8.5 Explainability

One of the key advantages of using ASP in the legal domain is its ability to provide clear and structured explanations for judicial reasoning. Once the ASP solver is run on a program P_{law} and a sentence C, the resulting stable model can be analysed to extract the reasoning behind the outcome. This feature makes the tool valuable for two main purposes. First, it can help explain the reasoning behind independent judicial decisions, offering a transparent view of how a verdict is reached. Second, it can generate different possible legal scenarios, which is particularly useful when dealing with vague legal concepts that may lead to different interpretations.

To achieve explainability, we propose two complementary approaches. The first approach relies on `xclingo`, a tool that allows rules and facts to be annotated in order to generate structured justifications. A key strength of `xclingo` is its ability to provide explanations for why certain atoms appear in the answer set, offering a detailed breakdown of the reasoning process. It also allows rules and facts to be described in natural language, making the explanation more accessible.

Example 8.5.1
Consider the following program, where Carlo has caused an injury to Beatrice:

```
cause("Carlo", "Beatrice", "skin lesion").
physical_illness("skin lesion").
intent_to_harm("Carlo", "Beatrice").
```

Using `xclingo` will produce a justification tree where each level of indentation represents a step in the reasoning chain leading to the conclusion. This structured approach makes it easier to trace how the decision was made. If we actually run `xclingo` we get:

```
Answer 1
*
|__It is evident that Carlo (perpetrator) caused injuries to Beatrice
↪  (victim)
| |__Carlo caused Beatrice to suffer skin lesion
| | |__skin lesion is an illness
| | | |__skin lesion is a physical illness
| | |__Carlo caused skin lesion to Beatrice
| |__Carlo had general intent to harm Beatrice
```

This result is obtained thanks to the annotations in the ASP code:

```
%!trace {"It is evident that % (perpetrator) caused injuries to %
↪  (victim)", X, Y} injuries(X, Y).
injuries(X, Y) :- cause_illness(X, Y), intent_to_harm(X, Y), agent(X),
                  agent(Y).

%!trace_rule {"% caused % to suffer %", X, Y, Z}
cause_illness(X, Y) :- cause(X, Y, Z), illness(Z), agent(X), agent(Y).

%!trace {"% is a physical illness", Z} physical_illness(Z).
illness(Z) :- physical_illness(Z).

%!trace {"% is an illness", Z} illness(Z).
%!trace {"% had general intent to harm %", X, Y} intent_to_harm(X, Y).
%!show_trace injuries(X, Y).
```

On average, explanations are generated in 0.08 seconds across 30 test cases. Interestingly, explanations for negative verdicts, *i.e.* where there was no evidence of the crime, took slightly longer, approximately 0.01 seconds more than those for positive ones.

The second approach, xASP, provides a more visual form of explanation by representing the answer set as a DAG. This graphical representation helps users intuitively understand the dependencies and logical steps involved in the legal reasoning process.

Example 8.5.2
In the judgment Cassazione penale sez. II, 21/02/2019, n.16899, the court classified as robbery the defendant's actions of holding the victim still by pressing her head while taking her earrings. This scenario can be encoded in ASP, assuming that the earrings were tightly secured to the earlobes, as follows:

```
own("Veronica", "earrings").
subtract("Giulio", "earrings").
snatch("Giulio", "earrings").
take_possession("Giulio", "earrings").
adherence("Veronica", "earrings", 4).
```

The resulting explanation, as visualised in 8.1 presents a graph where nodes represent both given and inferred facts, while arrows indicate the rules applied to derive conclusions. Each arrow highlights the dependencies between the facts, illustrating how the final outcome was reached.

Compared to `xclingo`, this approach is computationally more expensive, with an average execution time of 1.65 seconds across 30 test cases. Both techniques enhance explainability in different ways: `xclingo` provides structured, text-based justifications that are particularly useful for tracing detailed logical reasoning, while xASP offers a more intuitive visual representation of the decision-making process. By integrating both methods, we ensure that the legal reasoning behind the verdicts is not only transparent but also comprehensible to different types of users, from legal experts to those unfamiliar with formal logic.

8.6 Resources

As mentioned in the introduction of this chapter, the developed system is freely available at `http://clp.dimi.uniud.it/sw/`. Specifically, there are two resources that the reader can explore. The first, entitled *Encoding of legal vagueness in ASP*, contains the encoding of all the legal rules referenced in this thesis. The second resource, *Encoding of XAI LAW*, is instead a website designed to make the navigation easier among the various encodings. The homepage, see Fig. 8.2a, provides links to articles and judgments encoding, see Fig. 8.2b, both in ASP and ILASP. This enables users to read the ASP encoding alongside their corresponding textual (*i.e.*in natural language) version.

(a) Homepage of the website

(b) Judgments of the encoded articles

Figure 8.2: Website

The reader will find the examples of previous verdicts, the ones that have been encoded in ILASP, organized according to the articles to which they refer (Fig. 8.3).

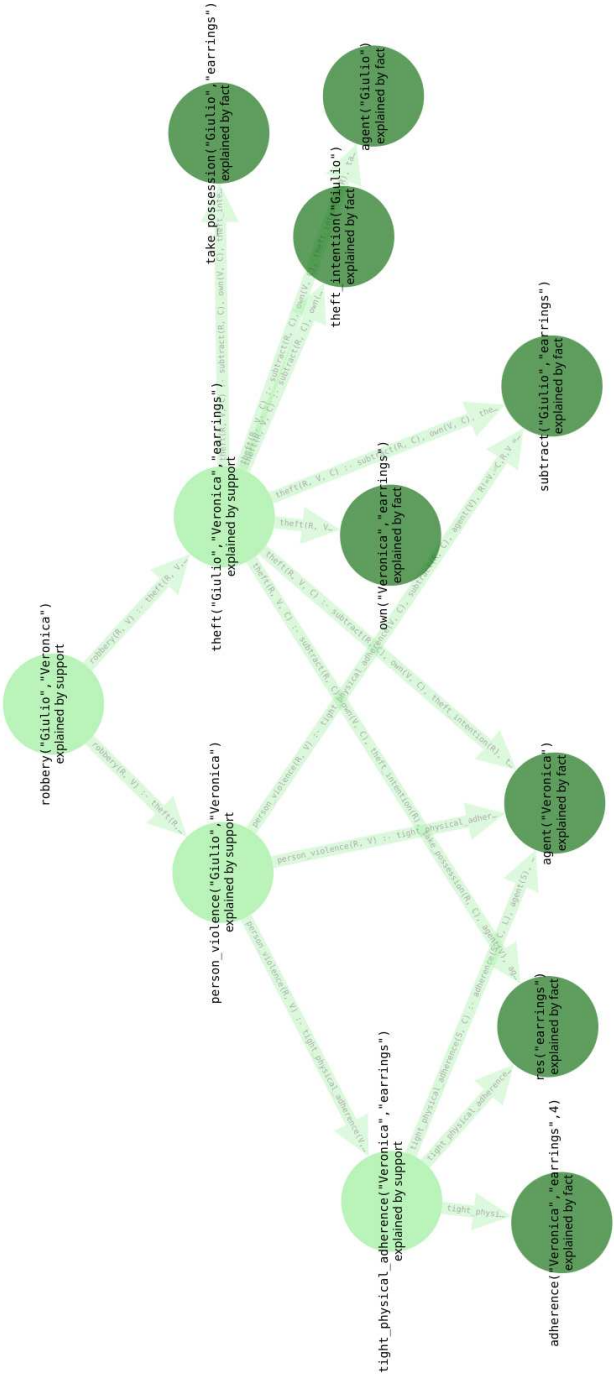


Figure 8.1: Explanation DAG made by xASP

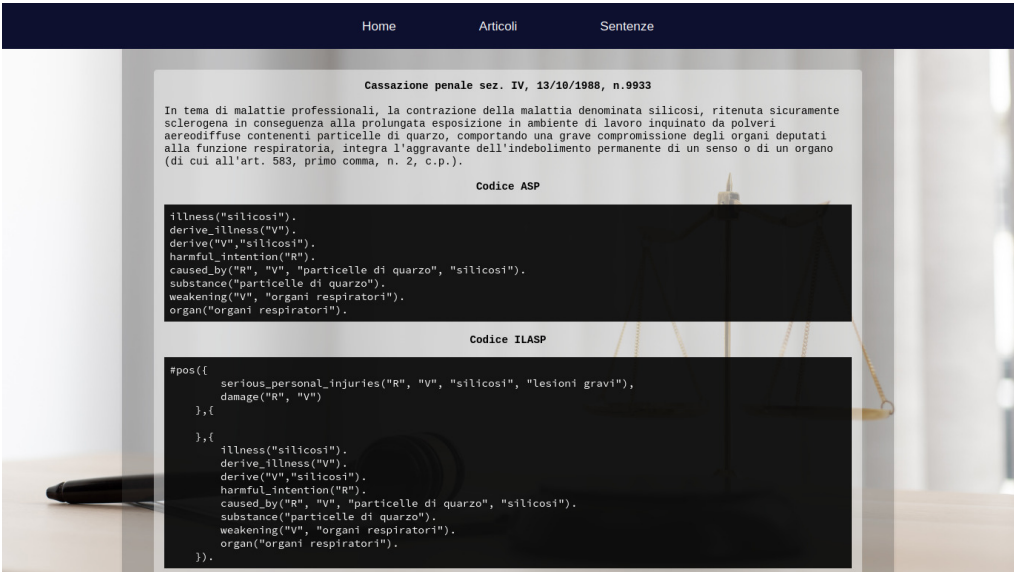


Figure 8.3: Example of a judgment with ASP and ILASP encoding

Furthermore, we developed a simple python and ASP-based application designed to serve as a decision support tool for simulating judicial reasoning. The application specifically focuses on modeling the first two levels of the Italian criminal justice system (*primo grado* and *appello*). The user interacts with the system through a structured questionnaire, as shown in Figure 8.4. Then, user's answers are automatically translated into an ASP program, effectively encoding the case facts as logic predicates. The encoded ASP program is processed by clingo: the stable model returned by clingo directly corresponds to the logically consistent final judgment of the case. The application proposes this outcome in natural language, as represented in Figure 8.5.

A core feature is the tool's ability to dynamically update facts and re-evaluate the case when new evidence is introduced. This capability is essential for simulating the fluid nature of a trial or the introduction of new material during the appeal stage.

8.7 Future Works and Conclusions

The current study focused on simulating the first two levels of the Italian criminal justice system (*primo grado* and *appello*) through an ASP-based tool capable of updating facts when new evidence is introduced. As future work, we plan to extend our framework to the third level (*cassazione*).

A notable advantage of adopting logic programming in this domain lies in its intrinsic explainability, which sets it apart from purely machine-learning-based techniques. ASP enables cautious and brave forms of reasoning by allowing judges (or legal experts) to inspect all stable models: properties that hold in all models can be considered robust, while those appearing in only some models may still inform provisional reasoning or suggest for further analysis.

```
Please, answer the following questions to get the final judgment.
We already assume that there was a theft (intentional) and that the stolen object was snatched.

[?] Choose the degree of judgment:
 > First degree
   Second degree

How many people are involved?
2

Insert their names:
1: Giulia
2: Rebecca

[?] What has been stolen: purse

[?] Who stole the object:
 > Giulia
   Rebecca

[?] Who was the owner of the object:
 > Rebecca

[?] Did thief make violence directly to victim to steal the object:
 > Yes
   No
   Not sure

[?] Did the thief make violence indirectly to the victim to steal the object:
   Yes
 > No
```

Figure 8.4: Some of the questions proposed by the application to generate an ASP program which solution corresponds to the judgment.

```
Final judgment: Giulia is guilty of robbery.
```

Figure 8.5: Judgment produced by the application in natural language.

Looking ahead, we aim to strengthen both the efficiency and the interpretability of our system. On the efficiency side, we will investigate the application of ASP parallelisation techniques [119, 120, 121, 122] to ILASP, with the goal of improving scalability when dealing with more complex or larger legal cases. On the explainability side, we will look into connections with argumentation-based approaches [123], thereby situating our results within a wider framework of computational models for legal reasoning.

Another key line of future work concerns knowledge acquisition. Although ILASP supports dynamic updates, the initial knowledge representation still relies heavily on human expertise for translating legal texts into ASP rules. This manual encoding process risks becoming a bottleneck as case complexity increases. To mitigate this, we plan to explore semi-automated or automated methods for extracting formal representations from legal documents, with the long-term goal of reducing reliance on manual encoding while maintaining legal accuracy and transparency.

9

Veterinary Medicine

The morphological evaluation of bull semen is a critical aspect of breeding soundness assessment, influencing fertility predictions and artificial insemination success. With the growing adoption of frozen semen and genomic selection, there is a pressing need for a standardized and automated approach able to analyze semen quality, particularly for the evaluation of spermatozoa abnormalities that affect semen freezing suitability and fertilizing capacity.

This chapter presents the work that was done to develop an AI-based system for sperm morphology classification, using YOLO networks. Exploiting an annotated image dataset and data augmentation techniques, the system identifies both normal and abnormal spermatozoa. This innovation promises to enhance the efficiency and accuracy of bull semen analysis, supporting advancements in reproductive technologies. The results are formalized in [124, 125, 126].

9.1 Related Works

The integration of advanced computational tools in the medical and veterinary fields has led to notable improvements in diagnoses' accuracy and efficiency [127, 128]. AI-based models, particularly artificial neural networks, have demonstrated effectiveness in predicting livestock parameters such as the weight of rabbits, poultry, pigs, and goats [129, 130, 131]. Similar approaches have been applied to estimate bull height [132] and to assess the economic feasibility of slaughtering bulls based on live weight prediction and milk yield in dairy cows [133]. Bull semen morphological evaluation has traditionally been conducted manually under a microscope, requiring considerable time and expertise, but, over the years, various technologies have been developed to improve this process. An example could be the *Computer-Assisted Sperm Analysis* (CASA) system [134], which has been validated for assessing sperm viability and motility in fresh semen samples. Unfortunately, this system, even if it has shown a noticeable performance, it is not able to classify morphological abnormalities. Furthermore, for the management of reproduction, there have been many studies [135, 136, 137] that have used AI to estimate semen quality

and ejaculate fertilizing capacity, not only in bulls. It has also been employed to predict the success of conception in dairy cows, the cleavage success of zygotes, and the *in vitro embryo production* (IVEP) [138].

9.2 Problem

The morphological evaluation of bull semen, an essential component of the *Bull Breeding Soundness Evaluation* (BBSE), is currently performed visually using bright-field microscopy. This process, which requires the expertise of trained personnel, is both time-consuming and subject to variability in results due to the reliance on human judgment. The evaluation is conducted in double by at least two distinct technicians, and the percentage of agreement should fall between 43% (Fair) and 58% (Moderate) to be considered sufficient to ensure the representativeness of the analysis [139]. The importance of these challenges arises from the fact that accurate semen analysis is critical for assessing sperm morphology, identifying abnormalities, and ensuring the suitability of semen for freezing and fertilisation. With the increasing adoption of artificial insemination and genomic selection schemes for young bulls, there is a growing demand for a more standardised and efficient method for sperm morphology evaluation.

9.3 Dataset

To the best of our knowledge, no database with a sufficient number of images exists to enable high-accuracy algorithms specifically for bull's spermatozoa morphology evaluation. This lack of data remains a significant barrier to the effective integration of such systems into veterinary practice. The dataset was therefore constructed *ad hoc* at the University of Udine and originally comprised 4890 microscopy-acquired fields. Each image contains 1 to 20 spermatozoa stained with eosin-nigrosine to distinguish alive from dead spermatozoa. Fields were captured using a phase-contrast microscope (40X magnification) equipped with a camera, using consistent settings for brightness (0.12), saturation (0.92), hue (0), and contrast (-0.13). Each spermatozoon was then manually boxed and annotated using *LabelImg* software. The dataset evaluation was performed by four expert operators, during routine semen analysis at the ANAPRI testing station (Fiume Veneto, Italy) as part of Italian Simmental young bull performance tests, as also shown in the study of Corte Pause et al. [140]. Categories included normal alive (NA), normal dead (ND), major abnormalities alive (MAA) or dead (MAD), and minor abnormalities alive (mAA) or dead (mAD). Fig. 9.1[1] illustrates the Society of Theriogenology's guidelines, highlighting how subtle the differences can be. Annotations were saved in YOLO-compatible text files, detailing object labels and bounding box coordinates for image processing.

As the reader can see, the BBSE guideline considers more than two categories for anomalies, but, since this was a first attempt to automatically classify bull's spermatozoa morphology, macro-categories were considered instead of the singular. This also makes the classification process easier as it is not required to distinguish between shapes that have very subtle differences.

[1] Image from `www.therio.org`

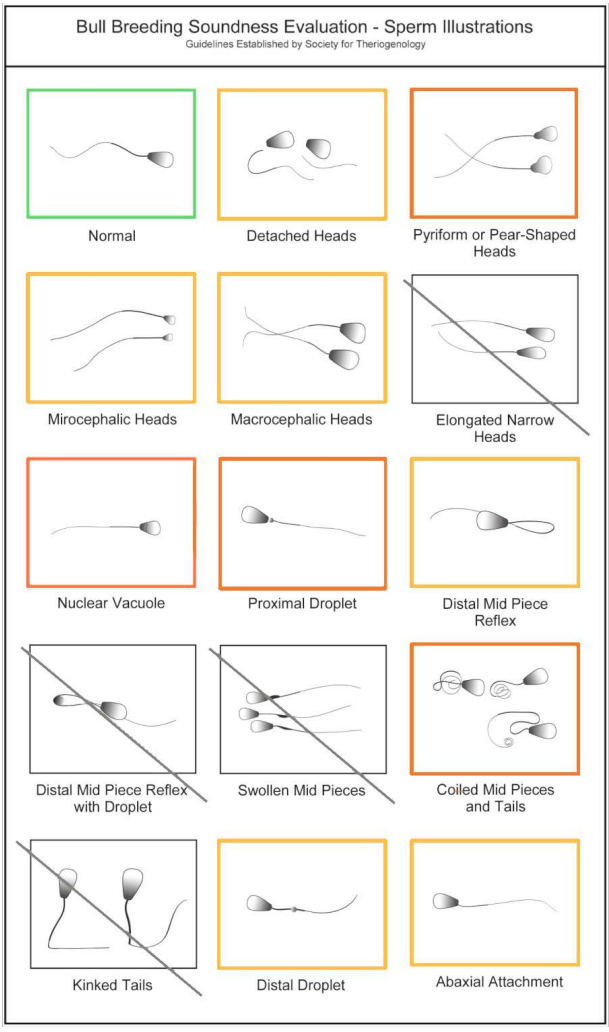


Figure 9.1: Guidelines established by the Society of Theriogenology

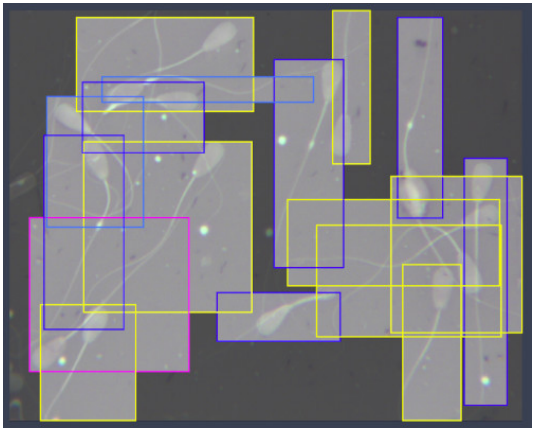
As shown in Fig. 9.1, spermatozoa are visually categorized using coloured boxes: normal cells are in green, those with major anomalies in orange, and those with minor anomalies in yellow. The ones that are crossed out were not considered. The possible anomalies are:

- **Major anomalies**: here we have two of the most difficult classes to identify using just computer vision. Pear-shaped heads, for instance, are highly subjective; despite being readily apparent to the human eye, a formal definition remains hard to catch. Nuclear vacuoles pose another difficulty for automated detection. These tiny structures are often indistinct and easily mistaken for image noise and *vice versa* by computer vision algorithms.

- **Minor anomalies**: it comprehends more cases than the previous. The primary challenge lies in detecting abaxial attachments, as the distinction from normal attachments can be very subtle.

On the other hand, Fig. 9.2 provides an example of a microscopy image with manually annotated bounding boxes, each colour-coded to represent different spermatozoon types. This part of the work was made by experts in the field of sperm evaluation. Alongside the image, it is displayed the corresponding XML-like file that encodes the class and bounding box information for each spermatozoon. Specifically, each line in the file represents a single spermatozoon: the first number indicates the class, while the subsequent values specify the bounding box coordinates and dimensions (`x_center, y_center, width, height`).

At the end of evaluation, the entire dataset was composed by a total of 26.915 manually boxed spermatozoa. The 56% of them were identified as NA, the 22% as ND, the 6% as MAA, the 4% as MAD, the 5% as mAA and the 7% as mAD. The low percentage of anomalies does not reflect the real average percentage of a generic sample, but is influenced by the contest for which it was sampled. Indeed, the slight predominance of NA spermatozoa over those that are dead and/or abnormal occurred because the semen analyzed was derived from bulls that were already approved by BBSE, which required a percentage of normal spermatozoa of at least 70%.



```
4 0.525000 0.746582 0.240625 0.118164
4 0.801172 0.261719 0.088281 0.488281
0 0.824219 0.809082 0.114062 0.379883
4 0.929297 0.661133 0.083594 0.601562
0 0.749219 0.565918 0.414062 0.209961
0 0.779297 0.659180 0.360156 0.271484
0 0.667188 0.187500 0.073438 0.373047
4 0.583984 0.373535 0.135156 0.506836
0 0.308594 0.527832 0.328125 0.415039
0 0.303125 0.131836 0.345313 0.228516
2 0.386719 0.192871 0.412500 0.061523
1 0.194531 0.692383 0.310937 0.375000
2 0.166797 0.369141 0.188281 0.318359
0 0.153125 0.857910 0.185938 0.282227
4 0.144922 0.541016 0.155469 0.472656
4 0.260937 0.260742 0.237500 0.171875
0 0.872266 0.594727 0.255469 0.380859
```

Figure 9.2: Spermatozoa image labelled

The dataset described above was employed to train the neural network for spermatozoa detection and classification. For the ILP module, a subset of these images was selected and subsequently converted into FastLAS example code, as detailed in Section 9.5.1.

9.4 NN Module

In this section, we describe the training process of the YOLO network for our specific task. As the primary objective of this work is the generation of meaningful explanations

rather than the optimisation of model performance, the training was conducted with the aim of obtaining a sufficiently accurate model, without dedicating extensive effort to maximising predictive accuracy.

9.4.1 Methods

Spermatozoa were classified following *Society for Theriogenology* (SFT) guidelines as normal or abnormal, and alive or dead, with abnormalities categorized into major and minor types. In addition, two augmentation techniques were employed to increase the number of samples. The *colour jittering* technique involves the random alteration of an image colour properties, including brightness, contrast, and saturation. This renders the model more resilient against lighting variations. This is particularly advantageous in real-life scenarios where the lighting conditions can vary. The second technique employed is *noise injection*, which involves the introduction of random noise into images with the objective of enhancing the model's capacity to tolerate imperfect or noisy data.

Once the augmented dataset was obtained, both the images and their corresponding text files were provided as input to YOLOv8 (YOLO, version 8; released in 2023) for processing. YOLO performed both the localization and classification of objects in a single unified step. Then, the entire image was processed by dividing it into a predefined grid, where each cell was responsible for predicting the bounding boxes of the potential objects within, as explained in Section 5.5.

Finally, the initial number of epochs was arbitrarily set to 100. However, an early-stopping function was implemented to stop training when the loss on the validation set ceased to decrease, which avoided overfitting and reduced the training time. If the model demonstrated a possible improvement after 100 training epochs, then it was trained for another 50 epochs. A series of tests revealed that the optimal result was achieved at approximately 100 epochs.

Accuracy, precision, and recall combine these elements to provide a holistic view of the model's performance. Accuracy provides an overall measure of the model's correctness, indicating how often the model is correct. However, accuracy is not an optimal metric for studying classification performance in cases in which positive instances of viability/morphology classification are significantly over- or under-represented in the training data.

As the model is trained to predict not only the class but also the bounding boxes, the performance is evaluated using mAP metrics, specifically mAP50 and mAP50-95. The AP is computed as the area under the precision-recall curve, providing a single metric that summarises the model's precision and recall performance. Instead, the mAP50 and mAP50-95 metrics incorporate different IoU thresholds (the ratio of the intersecting area to the total combined area of both boxes) to determine the accuracy of predicted bounding boxes relative to ground truth boxes.

9.4.2 Results

The following plots provide insight into the performance of our model. In particular, Figure 9.5 represents general metrics collected during training and validation. Figure 9.3 shows the precision-recall curve, which illustrates the typical trade-off between these two measures. It is particularly useful for evaluating performance on imbalanced datasets,

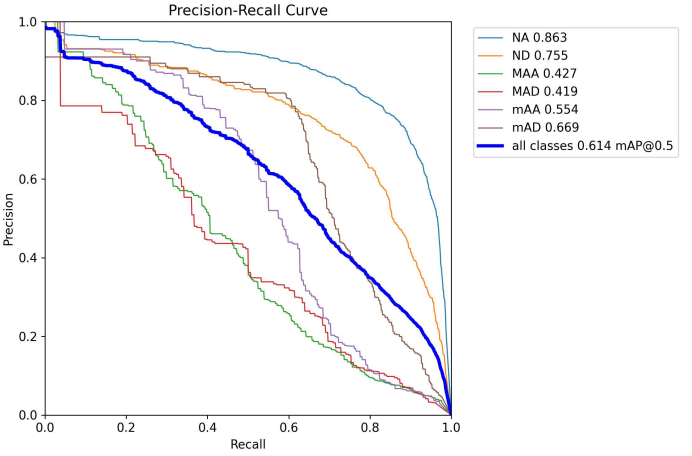


Figure 9.3: Precision-Recall curve

where one class is significantly more frequent than the other, as in our case. An effective model is characterised by a curve that approaches the upper-right corner, reflecting simultaneously high precision and high recall. In our results, the best performance is achieved for the NA class, whereas the MAA and MAD classes exhibit the lowest performance, with curves that are nearly overlapping.

The confusion matrix in Figure 9.4 provides a detailed view of the classification performance across the different spermatozoa categories. As expected, the NA class achieves the highest recognition rate, with 0.86 of correctly classified samples. This indicates that the model is able to reliably identify normal spermatozoa, which is consistent with the results observed in the precision–recall analysis. The ND class also performs relatively well, with 0.75 of instances correctly predicted.

By contrast, the performance on abnormal classes is notably lower. For example, MAA and MAD show a large proportion of misclassifications, with only 0.37 and 0.34 correctly classified, respectively. This suggests that the morphological differences between these categories are subtle and difficult to capture for the neural network. For the minor classes, such as mAA and mAD, the results are slightly better with accuracies of 0.52 and 0.66.

9.5 ILP Module

This section examines the FastLAS encoding of the dataset and its corresponding results. A detailed explanation of the background knowledge is not provided, as it merely defines the domain of the considered features.

9.5.1 Examples

To encode the examples for this study, a custom application, that we called *XAI Bull Spermatozoa Classification* (XAI-BSC), was developed to extract key morphological

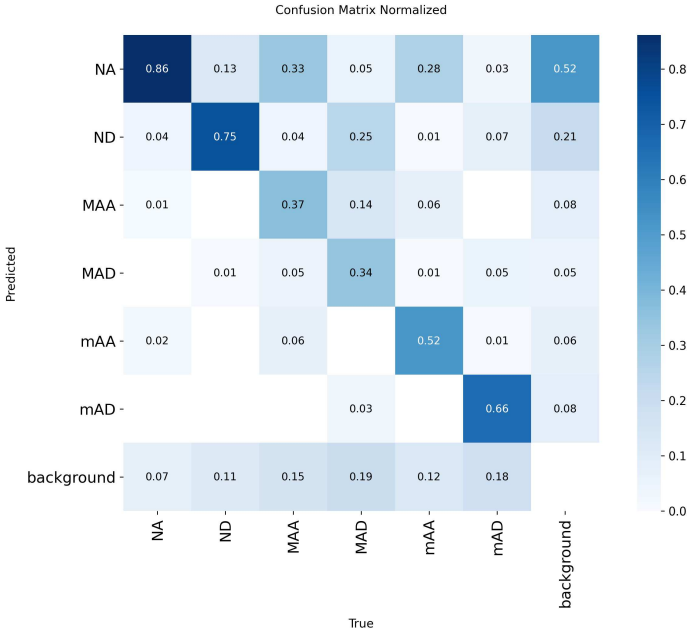


Figure 9.4: Normalized Confusion Matrix

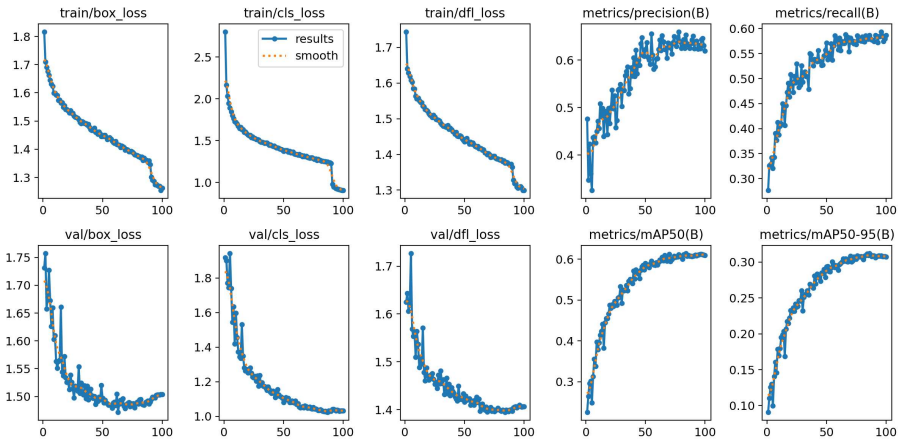


Figure 9.5: Metrics indicating the trend for the YOLO model performance during training and validation.

features from spermatozoa images. These extracted features were then formatted into a structured representation suitable for learning with FastLAS. Given that the objective of this study is to enhance the explainability of deep learning models through logic programming, the ILP module was built without incorporating machine learning or other black-box techniques. Instead, all morphological information was obtained using traditional computer vision methods.

The process begins with the application taking as input the manually annotated bounding boxes (specifically, the YOLO annotation file as in Figure 9.2) along with the corresponding image. It then presents each spermatozoon individually, prompting the user with a series of questions regarding its morphological traits. These questions help capture details that are currently difficult to detect automatically, such as tail curvature. In some cases, indirect inferences can be drawn from other features; for instance, the bounding box's aspect ratio can provide clues about tail morphology.

Following this, XAI-BSC attempts to delineate the contour of the sperm head, as its geometric properties are crucial for classification. A pear-shaped contour, for example, is often indicative of a major anomaly. Head detection is carried out in two stages: first, potential head regions are identified using the *Hough Circle Transform* (HCT); second, contour refinement is performed with *active contour models*. The HCT detects candidate regions by identifying circular patterns within the image based on their bright intensity and shape. This technique operates by applying the *Canny* edge detector to extract contours, after which it maps potential circles using an accumulator space. Within this space, *votes* represent degrees of radial symmetry, with higher symmetry corresponding to a stronger likelihood of a circle at that location. Finally, the peaks found in this accumulator space indicate the most probable circle centres and radii, which are then selected as candidate head regions.

Once candidate regions are selected, the system filters them based on size and polishes their contours using active contour models, also known as *snakes*. This method iteratively adjusts an initial contour based on image gradients and smoothness constraints, allowing it to converge onto the actual boundary of the sperm head. The initial contour is set as a circle three times the estimated head radius, which provides a starting point for the snake to converge onto the actual boundary. A Gaussian-smoothed version of the image is used to suppress noise while preserving key structural details. The final output is a clear contour that accurately follows the sperm head's boundary. However, due to the low contrast and almost grayscale nature of the images, contour detection can be challenging. Furthermore, in images containing multiple spermatozoa, it is not always straightforward to determine which contour corresponds to the target of interest.

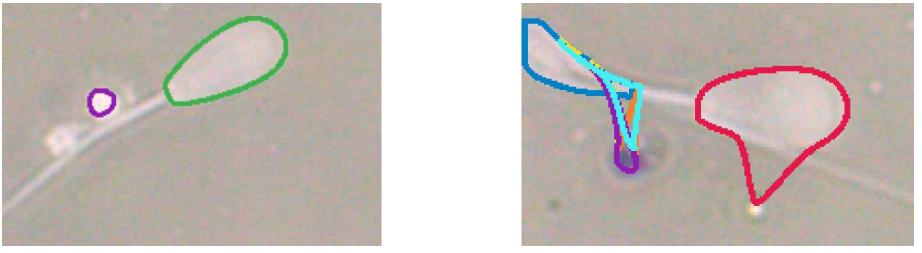
When the head contour cannot be reliably detected (see Fig. 9.6b), the system falls back on features derived solely from the bounding box. These include redness, bounding box aspect ratio, and area. Redness is particularly relevant for distinguishing between live and dead spermatozoa. The bounding box ratio provides information about potential morphological irregularities, such as whether the sperm has not a tail or if it appears coiled. Indeed, detecting the sperm tail presents a difficult challenge, as its shape varies significantly across different anomalies. However, by analyzing the aspect ratio of the bounding box surrounding the spermatozoon, it is possible to infer some characteristics of the tail. Specifically, a ratio almost equal to one suggests that the tail

is curled, coiled, or absent. The bounding box area offers further insight into the overall size of the sperm.

Conversely, when the contour is successfully extracted (see Fig. 9.6a), XAI-BSC computes additional features, such as the contour length, enclosed area (for precise head size measurement), and the ratio between the contour's minimum and maximum axes. These features provide a more detailed characterization of sperm shape. The system also calculates roundness, a measure of how closely the sperm head resembles a perfect circle, using the following formula:

$$Roundness = \frac{4 \times \pi \times A}{2p^2} \tag{9.1}$$

where A represents the enclosed area and $2p$ denotes the perimeter. This metric helps distinguish normal spermatozoa from those with morphological abnormalities, as mentioned earlier.



(a) Correct detection (green). (b) Wrong detection (any of them).

Figure 9.6: Examples of automatic contour detection of the head.

Regarding the features that are not currently extracted automatically, it is important to clarify the underlying reasons:

- The class of detached heads may appear to be easily detectable; however, in practice, it requires a robust algorithm capable of reliably identifying the presence of a tail. For instance, while we are able to detect the thickest segments of the tails, other parts may not be highlighted. If such missing segments occur near the junction with the head, see Fig. 9.7, it becomes particularly challenging to determine whether the head is actually detached or if the detection error is simply due to an incomplete tail representation. A similar problem arises in the case of abaxial attachment.

- Proximal and distal droplets could, in principle, be detected using the same technique employed for identifying the head, *i.e.* using HCT, filtering by smaller circular regions. However, the main issue is the presence of numerous artefacts in the images that visually resemble droplets, leading to frequent false positives (see Fig. 9.8).

- Pyriform heads are challenging to define, as there is no objective and universally accepted definition of what constitutes a pear shape. This classification typically

Figure 9.7: Detached tail close to head attachment

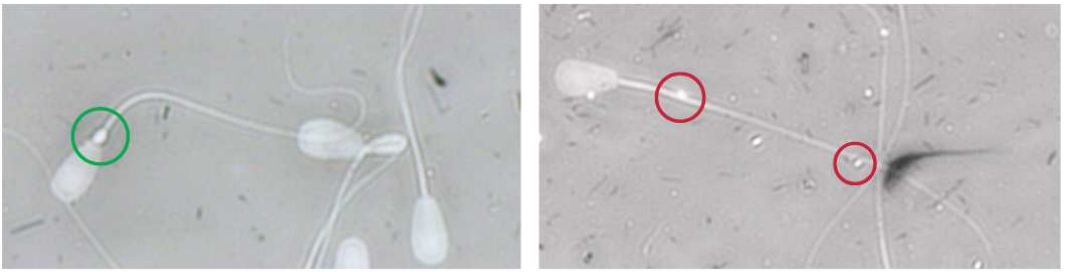


Figure 9.8: Real distal droplet (green, left) and artifacts (red, right)

requires human judgment or the use of machine learning techniques, even if the roundness might help somehow.

- Nuclear vacuoles are generally too small to be reliably detected in our current setup, as the microscope magnification used during image acquisition was insufficient for capturing such fine details.

- The detection of distal midpiece reflexes also depends on robust tail segmentation. When tails overlap or cross each other (Fig. 9.9), it becomes difficult to determine the correct structure and orientation, which can result in ambiguous or incorrect interpretations. Moreover, establishing the correspondence between each tail and its respective head is not straightforward. The same limitations apply to the detection of coiled midpieces and coiled tails.

We observed that performing two separate training sessions, rather than a single combined one, led to better results. The first session focused exclusively on determining spermatozoon viability, *i.e.*, whether it was alive or dead. This separation of viability and anomaly enabled the training process to achieve better performance by focusing on learning one feature at a time. This contrasts with the deep learning module, which attempts to infer all characteristics simultaneously.

Figure 9.9: Different tails crossing and overlapping each other.

For the first case, the example inclusions contained only the predicate `label` with `a` (alive) or `d` (dead). An example is illustrated as follows:

```
#pos(id3_12@100,
    {
        label("a")
    }, {
        label("d"), curledTail,
        missingTail, singleCoilTail, overtunedHead
    }, {
        red(648). area(25200). ratio(175).
        headRoundness(74). headArea(2594). lenghtHead(210).
        ratioHead(2). bubbleHead.
    }).
```

The second training session was dedicated to distinguishing among anomaly types: no anomalies (`n`), major anomalies (`M`), and minor anomalies (`m`). The format of examples in this session followed a similar structure:

```
#pos(id58_12@100,
    {
        label("M")
    }, {
        label("n"), label("m"), curledTail,
        missingTail, singleCoilTail, overtunedHead
    }, {
        red(648). area(25200). ratio(175).
        headRoundness(74). headArea(2594). lenghtHead(210).
        ratioHead(2). bubbleHead.
    }).
```

In both trainings, the example inclusions contained the label of the ground truth class, while exclusions comprised all other class labels, as well as any morphological characteristics that were not selected by the user within the app (in the case of the anomaly classification, that is the second training). The context for each example included all relevant characteristics, whether automatically extracted from the image or manually annotated through XAI-BSC.

One advantage of using FastLAS is its reduced dependence on large training datasets compared to models like YOLO. As a result, we were free to design a balanced dataset, assigning equal importance to each class. Specifically, we set the same weight (of 100) for all examples, as we did not have any prior knowledge about which examples were more representative or more likely to be noisy. If we had such information available, we could have adjusted the weights accordingly, for instance by assigning higher weights to more representative examples and lower weights to potentially noisy ones.

To enhance the generality of the ILP model, we adopted a 10-fold cross-validation approach (see Section 5.3.2). This method enabled us to incorporate a wider variety of examples, thus reducing the risk of overfitting and improving the robustness of the model. For each fold, the training set contained 12 spermatozoa per class, while the test set contained 7 spermatozoa per class.

9.5.2 Hypothesis Space and Bias

As explained in Chapter 3, FastLAS employs a function to guide the rule-learning process. In our work, we defined two distinct cost functions, reflecting the fact that different features are relevant depending on the classification objective (viability or anomalies).

For the viability classification task, we applied a strong bias to encourage the learning of general rules that cover multiple examples. The following penalties were used:

```
#bias("penalty(100, head(X)) :- in_head(X).").
#bias("penalty(1, body(X)) :- in_body(X).").
```

The meaning of this code is that, since we are assigning a penalty of 100 to the head of the rule and 1 to each body predicate, then covering a single example will lead to a minimum cost of 101 (one for the body and 100 for the head). This setup encourages the generation of rules that cover multiple examples simultaneously, since failing to explain an example already imposes a significant penalty of 100 (see Section 9.5.1). While the values 100 and 1 are arbitrary, the key idea is to use a large disparity to bias FastLAS toward generalisation.

The second function, the one for anomalies, is slightly more complex. Here, the penalty for the head is reduced to 10, to highlight that anomaly detection involves a wider and more diverse set of morphological features, and thus requires a more complex rule set. For the predicates in the rule body, we further refined the penalty scheme by categorising features into two groups:

- *class-specific* feature: these include features strongly indicative of particular anomalies. For example, a missing tail is characteristic of major anomalies, while a coiled tail is associated with minor anomalies. These features are both necessary and sufficient for determining the class, and thus receive the lowest penalty of 3.

- *general morphological* features: these features, such as the ratio of the box, are usually informative across multiple classes. Since they are helpful but less crucial, they receive slightly higher penalties. For instance, `head area`, `roundness`, and `ratio` are assigned a cost of 4. Less informative features, such as `redness` and `head length`, are penalised more (cost of 5), as they are more relevant for viability assessment or contribute less directly to anomaly detection.

Although the differences in penalties between features are relatively modest, they are crucial in balancing rule complexity and generalisability. This design ensures that the learned rules prioritise the most discriminative characteristics while reducing the risk of overfitting.

9.5.3 Results

Given the intrinsic differences between deep learning and logic-based approaches, it is unsurprising that FastLAS exhibits significantly lower performance compared to YOLO. However, this outcome aligns with expectations, as the primary objective of integrating FastLAS is not to surpass deep learning in predictive accuracy but rather to enhance explainability in decision-making. The rules generated by FastLAS usually follow a structured form, such as:

```
pred_label("m") :- distalDroplets, ratio(V_0_rr), V_0_rr >= 77.
```

To assess the effectiveness of the ILP approach, we conducted a comparative analysis between FastLAS and traditional machine learning models, including SVM, *Decision Trees*, and *Random Forests*. These models, while capable of learning from data, generally exhibit lower expressiveness and adaptability than deep learning systems. As illustrated in Fig. 9.10, FastLAS consistently outperforms these conventional machine learning techniques across multiple evaluation metrics, demonstrating its ability to extract meaningful and interpretable rules from the data.

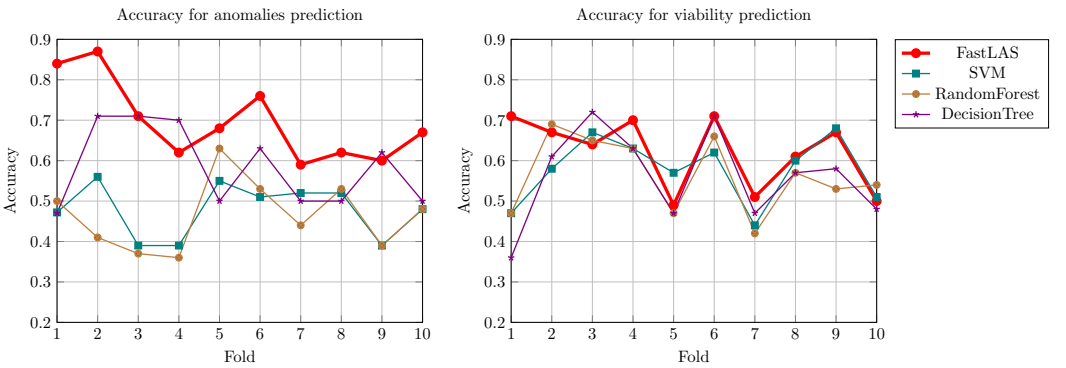


Figure 9.10: Accuracy for each fold for each model. On the left, the graph relative to prediction of anomalies and, on the right, the one relative to viability

Table 9.1 presents a comparative analysis of the models' performance in classifying spermatozoa into three categories: normal, major anomaly, and minor anomaly. The

Table 9.1: Performance comparison of different models for anomalies prediction.

	Precision_M	Recall_M	Precision_μ	Recall_μ	Accuracy	Weighted F1
FastLAS	0.70	0.69	0.78	0.79	0.70	0.73
SVM	0.33	0.48	0.44	0.48	0.48	0.40
Random Forest	0.46	0.46	0.45	0.46	0.46	0.46
Decision Tree	0.47	0.45	0.46	0.45	0.58	0.45

Table 9.2: Performance comparison of different models for viability prediction.

	Precision	Recall	Accuracy	Weighted F1
FastLAS	0.63	0.65	0.62	0.63
SVM	0.73	0.54	0.58	0.55
Random Forest	0.64	0.54	0.48	0.40
Decision Tree	0.52	0.55	0.56	0.56

evaluation metrics are reported using both macro-averaging (M) and micro-averaging (μ), as detailed in Section 5.3.3 (by the equations 5.1, 5.2, 5.3 and 5.4).

FastLAS achieves an accuracy of 70% and a weighted F1-score of 0.73, demonstrating a reasonable balance between precision and recall. The higher recall of 0.79 in the micro-averaged results suggests that the system effectively identifies most instances of anomalies, although this may come at the cost of reduced precision.

For the classification of spermatozoa as either alive or dead, the results are summarised in Table 9.2. FastLAS attains an accuracy of 62% and a weighted F1-score of 0.63, indicating a moderate ability to distinguish between live and dead spermatozoa. The recall value of 0.65 suggests that the system correctly identifies a considerable proportion of positive cases, though further refinements may enhance its overall predictive performance.

Overall, although FastLAS exhibits lower performance compared to YOLO, it remains competitive with traditional machine learning models while providing the crucial advantage of explainability. This aligns with the primary objective of this study, which prioritises interpretable decision-making over raw predictive performance.

9.6 Explainability

As outlined at the beginning of this thesis, our primary objective is to develop systems capable of explaining the predictions made by the deep learning module. While the rules learned by FastLAS are highly interpretable, they alone do not reconstruct the full reasoning process that is the specific sequence of rule applications leading to a given prediction. To address this, we exploit xASP2. Specifically, in XAI-BSC we have employed an interactive part that integrates YOLO with ASP.

9.6.1 System Description

The system workflow begins with a user uploading an image, which is then processed by YOLO to detect and classify spermatozoa. The detected objects, along with their

predicted labels and bounding boxes, are displayed to the user, who can select a specific spermatozoon for further explanation. Once selected, the spermatozoon is shown in isolation using the sub-image extracted from the bounding box. As in the dataset preparation for FastLAS, morphological features are partially provided by the user and partially extracted automatically. These features are then encoded into ASP facts following the same schema used during rule learning. The feature encoding, along with the rules learned by FastLAS, is provided as input to `clingo`, which computes the stable models corresponding to the classification process.

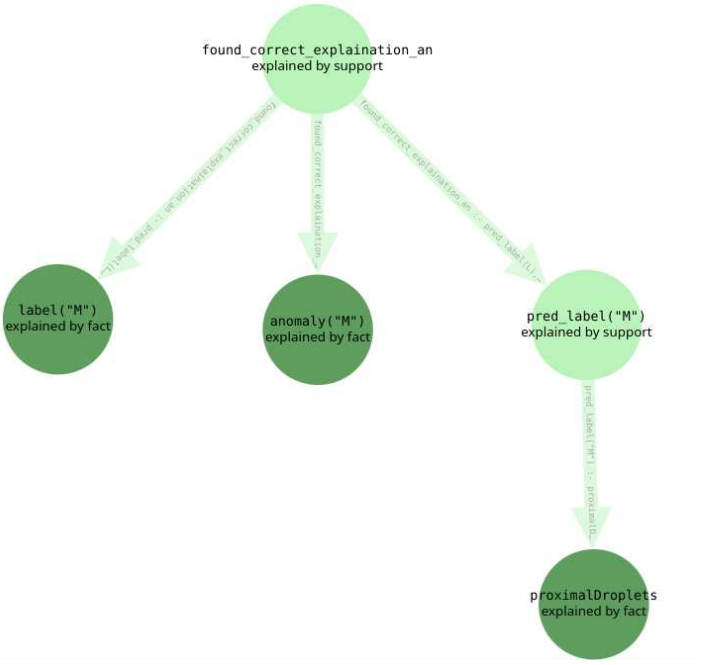
Using xASP2, we generate a directed acyclic graph that visually represents the reasoning steps leading to the final classification. This enables us to trace the logical inference path supporting the model's decision. It is important to note that the ASP-based classification does not always align with YOLO's predictions. In some cases, no stable model is found, or the rules yield an incorrect classification. To ensure that explanations correspond specifically to the YOLO prediction, we introduce an auxiliary predicate (`found_correct_explanation`), which holds true only when the ASP-derived label matches YOLO's output. All other ASP-derived classifications are disregarded. Once the correct reasoning path is identified, it is translated into natural language to provide an intuitive explanation to the user. This process involves traversing the DAG generated by xASP2, extracting variable values from feature nodes, and substituting them into pre-defined explanation templates.

Figures 9.11 and 9.12 illustrate the explanations generated by the system. Fig. 9.11 presents an example where a spermatozoon is classified as having a major anomaly. The explanation highlights the presence of a proximal droplet as the key morphological feature justifying this classification. The information is represented both in natural language and through an explanation graph. In contrast, Fig. 9.12 shows an explanation for a spermatozoon classified as normal. Here, the system supports its prediction by emphasizing the absence of specific abnormalities (*e.g.*, bent tail, bent neck, and proximal droplets), represented as red nodes in the graph. Additionally, the system incorporates numerical features such as the ratio value (0.37, scaled for the ASP encoding) to reinforce the classification decision.

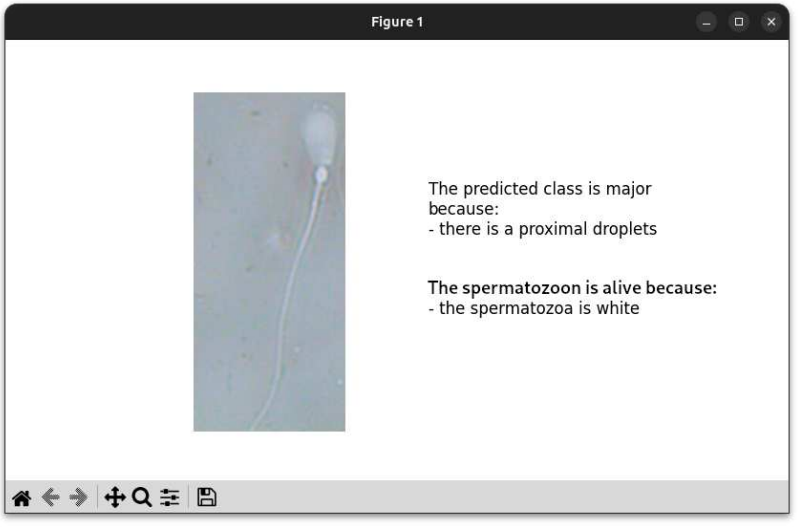
9.6.2 Comparison with LIME and SHAP

To better assess the strengths and limitations of our proposed approach, we also experimented with two of the most widely adopted post-hoc explanation methods, LIME and SHAP (see Section 4.4.5).

When applied to the spermatozoa recognition task, LIME produced largely unsatisfactory results. Its main advantage was efficiency, requiring on average about one minute per image, which is significantly faster than SHAP. However, the explanations were of limited utility: the highlighted regions rarely corresponded to meaningful biological structures and were therefore difficult to interpret and unreliable for domain experts. In most cases, LIME focused on arbitrary parts of the background rather than the spermatozoon itself, and even when spermatozoa regions were highlighted, they typically did not correspond to features relevant for classification. An illustrative example is provided in Figure 9.13, where the yellow contour marks the region identified by LIME as most influential for YOLO's prediction.



(a) xASP2 DAG for the anomaly prediction in Fig. 9.11b.



(b) Image of spermatozoa with explanation of anomaly and viability.

Figure 9.11: DAG and its translation in natural language

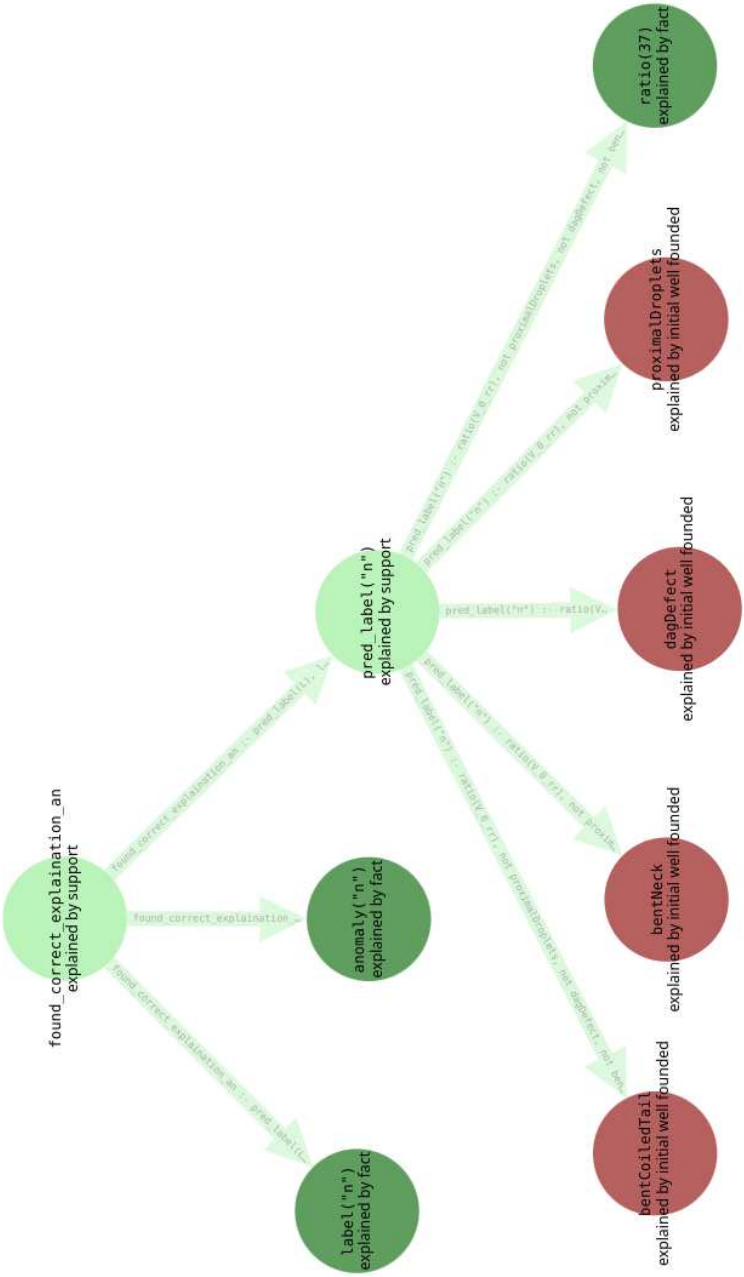


Figure 9.12: xASP2 DAG for a prediction of no anomalies (normal)

Figure 9.13: LIME explanation.

Furthermore, because the perturbed images are generated randomly, applying LIME multiple times to the same instance produces highly variable results, undermining its reliability. The example shown in Fig. 9.13 represents one of the best outcomes we were able to obtain.

To address the variability observed with LIME, we also conducted explainability tests using SHAP. To enable SHAP analysis without exhausting memory, we downsized the original images, arbitrarily setting the larger dimension to 100 pixels. This reduction inevitably led to a loss of important visual details, resulting in coarse explanations that were difficult to interpret. Even after resizing, processing each image still required an average of 14 minutes. An example of the output is shown in Figure 9.14, where pixel colours indicate their contribution to the model's prediction: green denotes positive contributions, purple negative contributions, and white pixels are neutral.

The results proved to be of limited interpretability. As illustrated in Figure 9.14, SHAP highlights pixels without clarifying the underlying morphological cause. For instance, the anomaly in this spermatozoon is the coiled tail; however, the explanation focuses on the proximal tail segment near the head, which could correspond to a different type of anomaly. This ambiguity prevents determining whether the highlighted region corresponds to the actual structural defect (the coil) or to a nearby, unrelated segment of the tail, thereby failing to provide a truly interpretable explanation.

9.6.3 Comparison with LLMs

LLM represent a different paradigm for explainability, as they generate natural-language justifications for a model's predictions rather than relying on explicit numerical feature attributions or perturbation-based analyses. The main idea behind using LLMs for explainability is to exploit their ability to interpret input–output pairs and produce textual explanations that are intuitive and human-friendly. However, LLM-based explanations

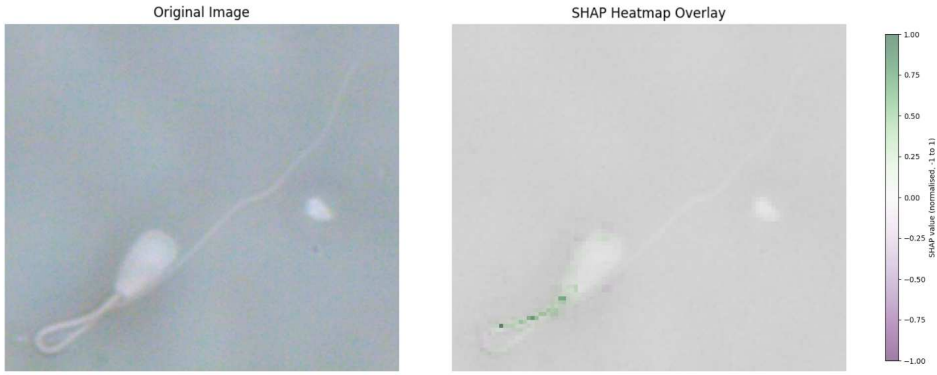


Figure 9.14: SHAP explanation.

are generated via probabilistic language modelling: the model predicts the next word in a sequence by assigning probabilities based on patterns learned from vast amounts of text. Consequently, while the explanations may appear convincing, they are not guaranteed to faithfully reflect the actual decision-making process of the black-box model. This lack of formal guarantees limits their reliability in high-risk domains, since there is no assurance that the explanation truly corresponds to the causal factors behind a prediction. Furthermore, LLMs themselves are black-box models, so using them to explain another opaque system effectively means employing one black box to interpret another, which exacerbates the trustworthiness issue. In this study, we experimented with two widely adopted LLMs, namely ChatGPT (version 5) and Gemini (version 2.5 Flash).

To enable a comparison with LLM-based explanations, we collected textual explanations from ChatGPT, Gemini, and XAI-BSC, which were then presented to domain experts along with a survey to gather their evaluations. Due to the difficulty of recruiting specialists in this field, feedback was obtained from only three experts at this stage. The survey assessed each explanation in terms of accuracy (the correctness of the explanation), completeness (the extent to which it covers all relevant factors), reliability (the degree of trust in the explanation), clarity (ease of understanding), and precision (the correctness of medical or biological statements). Experts rated each characteristic using a five-point Likert scale, where 1 indicates the lowest quality and 5 the highest. The correctness ratings for the LLM explanations were low, with mean scores of 1.33 out of 5 for Gemini and 1.67 out of 5 for ChatGPT. For XAI-BSC, the first explanation obtained a similarly low score (1.33 out of 5), whereas the second one was rated maximally by all participants (5 out of 5), illustrating the added value of offering alternative reasoning paths. Indeed, another advantageous feature of XAI-BSC is its ability to produce multiple answer sets when running ASP on the image encoding. As a result, the same background knowledge can lead to distinct, but equally valid, explanations. This multiplicity of reasoning paths offers a wider perspective on the decision.

9.7 Future Works and Conclusions

In this work, we integrated YOLO for object detection with FastLAS for rule learning, and presented XAI-BSC, a system capable of achieving both accurate classification and interpretable explanations. While the approach proved effective, we also identified some limitations in using FastLAS, particularly in terms of scalability when the number of features, examples, or the size of the hypothesis space increases. To mitigate this issue, we restricted the learning to separate sub-tasks (viability and anomaly classification).

Looking ahead, several directions for improvement and extension are envisioned. First, we aim to enhance the accuracy of sperm head detection, which, although currently effective, still leaves room for improvement. Beyond this, we plan to develop reliable methods for tail detection and to automate the identification of additional morphological features, such as proximal droplets. These enhancements will reduce the reliance on manual input, increase the autonomy of the system, and broaden the scope of explainable morphological analysis in bull sperm classification.

10

Logical Reasoning and Teaching

Logical reasoning is fundamental to succeed in scientific discipline, programming, and problem-solving. However, admission tests, such as the TOLC-S, indicate that the majority of students struggle when confronted with questions requiring logical inference or abstract thinking. The issue is addressed by the development of TOLC-ASP, which is a logic-based educational tool designed to assist students in improving their reasoning skills. It is based on ASP and it is designed to generate, execute, and explain logical quizzes, similar to the ones of the TOLC-S. It aims to support students in improving their reasoning abilities through practice, feedback, and logically grounded explanations. It was presented in [141].

10.1 Related Works

ASP is particularly well-suited for encoding and solving logical puzzles when formulated correctly [142, 143, 144]. Solutions are derived and justified through formal logic and rigorous deduction under the stable model semantics. Moreover, explanations can be provided to both experts and non-experts thanks to other tools developed for this purpose [51, 54]. There has been, and still is, an important interest in teaching logic programming, both as a means of promoting computational thinking and as an introduction to declarative problem solving. Experiences in high schools have shown that ASP can be effectively used for teaching puzzle encoding and solving, even without having been previously introduced with other courses [145]. Similar initiatives have also been provided using Prolog, which has a long tradition in education and has been widely employed for introducing students to logic-based reasoning and declarative programming [146, 147, 148].

10.2 Problem

The CISIA Online Test[1] (TOLC) is an admission test developed by the *Italian Inter-university Consortium of Integrated Access Systems* (CISIA). It is widely adopted by Italian universities as a means to verify the minimum level of knowledge required for access to undergraduate programmes, as well as to guide students towards the most appropriate study path.

Different variants of the TOLC correspond to specific academic disciplines. In this work, the focus is on the TOLC-S, designed for scientific degree programmes. The test consists of 55 questions divided into six categories: 20 in Mathematics, 15 on "Reasoning, Problems and Text Comprehension", and 5 each in Biology, Chemistry, Physics and Earth Sciences. Of these, maths and reasoning-based questions represent the domains in which students typically face the greatest difficulties, as confirmed by official reports [149] (see Table 10.1). Before 2024 the test had a slightly different structure, including 20 questions on Mathematics, 10 on Reasoning and Problems, 10 on Text Comprehension and 10 on Sciences.

		2021		**2022**		**2023**	
	n	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
Total	50	22,0	11,5	21,8	11,5	21,1	11,5
Mathematics	20	10,0	5,9	10,0	5,8	9,5	5,8
Reasoning and Problems	10	3,5	2,5	3,3	2,5	3,3	2,5
Text Comprehension	10	4,8	2,7	4,8	2,7	4,8	2,7
Sciences	10	3,6	2,6	3,6	2,7	3,5	2,7

Table 10.1: The data includes the mean and standard deviation (Std Dev) of scores. n represents the number of items/questions for each section.

LLMs have recently been proposed as potential tools to support students in their preparation for logic-related tasks. However, often these systems are not able to make deductive reasoning: instead they rely on statistical correlations learned during training. As a result, their explanations may appear plausible but are not always logically consistent or verifiable. ASP, on the other hand, provides a formal and rigorous computational framework for representing and solving logical puzzles, as the ones of TOLC.

10.3 ASP Encoding

This section presents the ASP encoding of logical (propositional/first order) puzzles, illustrating how to verify the existence of a unique solution while making explicit their underlying logical structure.

[1] `https://testcisia.it/studenti_tolc/login_sso.php`

10.3.1 TOLC Puzzles

An important preparatory tool for students facing the TOLC-S is the Mentor [150], a self-learning resource provided by CISIA. The guide is specifically designed to help students practise multiple-choice logic questions, although it does not aim to be a strict simulation of the official test since its purpose is just to familiarise learners with typical logical reasoning tasks.

The guide adopts an interactive format. Each exercise begins with the presentation of a question, after which the student selects one of the proposed answers. Depending on the choice, the guide redirects the student to a different page: if the answer is correct, a confirmation message is shown together with a new puzzle to solve; if the answer is incorrect, targeted feedback is provided and the student is taken back to the previous question. This structure allows students to also reflect on mistakes.

The Mentor includes a variety of logical puzzles, ranging from propositional logic to visual reasoning tasks. They do not require advanced mathematical knowledge, and all texts are formulated in plain natural language (Italian). Nevertheless, in order to succeed the students need to read puzzles carefully, as subtle linguistic cues can alter the logical meaning of a statement.

Logical puzzle included in the Mentor can be classified into:

- logical (propositional/first order) puzzles,
- puzzles that require analysis of a picture or a graphical representation to obtain the solution,
- puzzles that require to complete a series of numbers or similar,
- combinatorial puzzles.

For the development of the application, attention is restricted to the first category of puzzles, 32 out of the 51 included in the Mentor. These can be further classified into two categories:

- puzzles that involve or omit the use of quantifiers,
- puzzles where the task is either to select exactly one correct answer or to exclude exactly one option, the latter being referred to as *dual-answer* puzzles.

10.3.2 Encoding

The fundamental principles of the encoding process we propose are outlined as follows. Each puzzle requires the specification of certain domain-dependent properties (*e.g.* `person(a).` `person(b).` `brother(a,b).`). These elements vary from one puzzle to another and presuppose a basic familiarity with logical modelling. However, such domain specifications represent the relatively straightforward aspect of the task. The greater complexity arises from the logical structure of the puzzles themselves. Usually, they involve properties whose truth values must be determined based on a sequence of hints or constraints. Furthermore, the possible answers presented to the student are often expressed as Boolean or first-order formulas, from which a single correct option must be identified.

We will now list few best practice for encoding these problems within ASP, with the aim of ensuring both correctness and explanatory clarity:

1. *Non-deterministic choice.*
   If `p` is one of the properties of interest, a choice rule can be written as

   ```
   {p}.
   ```

   All properties whose truth value is not predetermined can be introduced using a non-deterministic choice rule. In cases where the available hints allow one to immediately conclude that p holds, this can be enforced directly by asserting as a fact

   ```
   p.
   ```

   If, instead, it can be immediately inferred that `p` cannot hold, this can be imposed adding a denial

   ```
   :- p.
   ```

2. *Disjunctive Hints.*

   A sentence such as *at least one of the following k literals is true* (in other words, their disjunction is true) can be expressed as:

   ```
   h :- ell_1.
   h :- ell_2.
   ...
   h :- ell_k.
   :- not h.
   ```

   For the sake of clarity, in the remainder of this section the symbol `h` will be used as a placeholder in the examples. In practice, however, each hint must be assigned a distinct predicate name.

   We can generalize this encoding to sentences such as "*At least m and at most n of the following k atoms/literals are true*", as follows:

   ```
   h :- m{ ell_1; ell_2; ... ;ell_k }n.
   :- not h.
   ```

3. *Conjunctive Hints.*
   If a hint specifies that a conjunction of positive or negative literals must simultaneously hold, this can be encoded as follows:

```
h :- ell_1, ell_2, ... ,ell_k.
:- not h.
```

4. *Boolean combinations.*
   More complex Boolean combinations can be constructed by combining the previously introduced patterns. For example, an implication provided as a hint can be encoded through its disjunctive semantics. Specifically, $(A \rightarrow B)$ can be reformulated as the condition that it is impossible for A to hold while B does not:

```
h :- A, not B.
:- not h.
```

5. *Quantifiers.*

   If `p` is a property (an atom) depending on a parameter x, a hint such as $\exists x\, p(x)$ can be forced as

```
h :- p(X).
:- not h.
```

   Similarly, a hint such as $\forall x\, p(x)$ can be expressed as:

```
nh :- not p(X).
:- nh.
```

   In this formulation, the predicate `nh` represents the negation of `h`, and the program enforces that such a predicate cannot occur. Intuitively, this corresponds to stating that there does not exist any x for which `p(x)` is false; equivalently, `p(x)` must hold for all x.

We will report an example below of an encoding of a quiz from the Mentor.

Example 10.3.1
Let us analyze the test n. 6 of the Mentor:

There are two brothers, Romolo and Remo.
One is always sincere and, conversely, the other is always a liar.
If Romolo says that their mother is named Silvia and Remo says that she is blonde, it can be deduced with certainty that:

A. if their mum is blonde, her name is not Silvia
B. if the name of their mum is Silvia, then her hair is blonde
C. the name of their mother is Silvia and she is not blonde

D. if the name of the mother is not Silvia, then she is not blonde
E. their mother is not called Silvia

The example can be encoded in ASP as follows:

```
%%% The two brothers
bro(romolo). bro(remo).
%%% Property: being sincere.
{sincere(X)} :- bro(X).
%%% Exactly one is sincere
h1 :- 1 { sincere(romolo); sincere(remo) } 1.
:- not h1.
```

After the first part we have to encode the fact that Romulus is sincere if and only if her mother is named Silvia (similarly for Remo and blonde).

```
%%% romolo is sincere iff the mother is Silvia
silvia :- sincere(romolo).
:- silvia, not sincere(romolo).
%%% similarly for remo
blonde :- sincere(remo).
:- blonde, not sincere(remo).
```

Then we have to encode the possible answers, assigning them a parameter.

```
%% A. The implication (blonde -> not silvia)
%%    is viewed as  not blonde or not silvia
opt(a) :- not blonde.
opt(a) :- not silvia.
%% B. (silvia -> blonde)
opt(b) :- not silvia.
opt(b) :- blonde.
%% C. silvia and not blonde
opt(c) :- silvia, not blonde.
%% D. (not silvia -> not blonde)
opt(d) :- silvia.
opt(d) :- not blonde.
%% E. not silvia
opt(e) :- not silvia.
```

Running the code with clingo one realizes that `opt(a)` is the unique option holding in all the (two) stable models (thus, it is a logical consequence under the stable model semantics).

10.4 TOLC-ASP

The implementation described in this section builds upon a collection of twelve quizzes selected from the Mentor, specifically exercises numbered 1, 6, 7, 20, 21, 23, 32, 41, 42, 48, and 49. In addition, an extra quiz, referred to as *quiz 0*, was incorporated from an example freely available on the CISIA website. Among these, quizzes 0, 1, 6, 7, and 42 do not involve quantifiers, whereas the remaining ones explicitly require their use. Notably, quiz 23 is distinguished by demanding a dual-answer formulation: the task is to exclude the wrong option rather than selecting the correct one.

The twelve selected quizzes were encoded in ASP following the methodology outlined in Section 10.3.2. Their solutions were obtained using the clingo solver, with the option 0 enabled to enumerate all stable models. This approach made it possible to determine whether a quiz was subject to specific constraints and, in all cases, to verify whether a candidate answer held across every stable model.

10.4.1 Quiz Parametrisation

Based on the previously encoded quizzes, additional variants were devised and their corresponding encodings systematically verified. Particular attention was dedicated to the introduction of parametrisation mechanisms, enabling the modification of elements such as names, actions, and attributes. To this end, a collection of placeholders was defined and associated datasets were constructed to provide substitution values (*e.g.* alternative human names). In this manner, starting from a limited set of predefined models together with auxiliary data files specifying names, actions, and related elements, it becomes possible to generate a combinatorially large number of distinct quizzes. This is achieved by instantiating the placeholders through the use of a random generator.

The following example demonstrates the transformation of Mentor's test no. 6 (see Example 10.3.1) into a generalised template and its subsequent encoding.

Example 10.4.1
The text of the question was firstly modified as follows:

*There are two people, *name1* and *name2*. One is always sincere and, conversely, the other is always a liar. If *name1* says that *sentence1* and *name2* says that *sentence2*, it can be deduced with certainty that:*
*A. if *sentence2*, then it's not true that *sentence1*.*
*B. if *sentence1*, then it's true that *sentence2**
*C. it's true that *sentence1* and it's not true that *sentence2**
*D. if it's not true that *sentence1*, then it's not true that *sentence2**
*E. it's not true that *sentence1**

While the placeholders **name1*, **name2** are self-explanatory, **sentence1** and **sentence2** denote simple propositions of the form subject–verb–complement (*e.g.* Adam eats an apple or Eva tricks the snake). The corresponding ASP encoding is presented below:

```
%%% The two people
person(name1). person(name2).
%%% Property: being sincere.
{sincere(X)} :- person(X).
%%% Exactly one is sincere
h1 :- 1 { sincere(name1); sincere(name2) } 1.
:- not h1.
%%% name1 is sincere iff sentence1
s1 :- sincere(name1).
:- s1, not sincere(name1).
%%% name2 is sincere iff sentence2
s2 :- sincere(name2).
:- s2, not sincere(name2).
%%% OPTIONS %%%
%% A. sentence2 -> not sentence1
opt(a) :- not s2.
opt(a) :- not s1.
%% B. sentence1 -> sentence2
opt(b) :- not s1.
opt(b) :- s2.
%% C. sentence1 and not sentence2
opt(c) :- s1, not s2.
%%% D. not sentence1 -> not sentence2
opt(d) :- s1.
opt(d) :- not s2.
%%% E. not sentence1
opt(e) :- not s1.
```

An evaluation of the readability of the generated quizzes was carried out. Since the texts are in Italian, the use of the conjunctive tense introduces an additional challenge, requiring further improvement to ensure clarity and fluency. In principle, this task could be supported by a LLM. An initial attempt was made to instantiate the quiz templates with real data using the Minerva LLM [151], developed by Sapienza NLP in collaboration with Future Artificial Intelligence Research (FAIR) and CINECA. However, when prompted to replace a template's placeholders with fabricated data, the model failed to deliver the expected result and instead reproduced the original template unchanged. As Minerva is under rapid development, further attempts will be conducted in the near future.

10.4.2 Feedback and Error Correction

For each test, designed in a parametric way, additional sentences were included to provide detailed feedback for wrong answers given by users. Following the approach adopted

in Mentor, the aim was to design a system which not only identifies the correct solution but also explains why an incorrect solution is incorrect. This method is designed to promote active learning, enabling users to recognise their mistakes and develop their logical reasoning skills. Implementing this feedback directly within ASP provided the advantage of enabling automatic verification of its logical soundness: by incorporating the ASP rules corresponding to the feedback for incorrect answers and automatically uncommenting them one at a time, the output produced by clingo explicitly displayed counterexamples to the respective answer options, thereby demonstrating their invalidity.

10.4.3 TOLC-ASP App

To support both the automated generation of quizzes and their execution, a dedicated Python program was developed. The system relies on a random generator to instantiate the tests and sequentially presents them to the user. For each quiz, the user can select a response and immediately obtain feedback. The design allows for an unrestricted number of attempts, ensuring that the same puzzle can be revisited repeatedly until the correct solution is discovered.

The program further provides a configurable option through the command-line parameter `--feedback`, which determines whether hints are displayed following incorrect responses. Question templates are maintained in JSON format and contain placeholders that are dynamically instantiated with values retrieved from a supplementary dictionary. This dictionary includes collections of names, actions, attributes, and other relevant elements. As a concrete example, the representation of Mentor's test no. 6, discussed earlier, is reported below.

```
{"id": 3,
 "source": 6,
 "text": "There are two people, *name1* and *name2*.\n
     One is always sincere and, conversely, the other is always a liar.\n
     If *name1* says that *sentence1* and *name2* says that *sentence2*,
     it can be deduced with certainty that:",
 "options": {
     "A": "if *sentence2*, then it's not true that *sentence1*",
     "B": "if *sentence1*, then it's true that *sentence2*",
     "C": "it's true that *sentence1* and it's not true that *sentence2*",
     "D": "if it's not true that *sentence1*, then it's not true that
          *sentence2*",
     "E": "it's not true that *sentence1*"
   },
   "exact": "A",
   "feedback":{
     "A": "Correct!",
     "B": "Wrong! *name1* and *name2* cannot be sincere at the same time",
     "C": "Wrong! There is a case where *name2* tells the truth and
```

```
            *name1* tells a lie",
        "D": "Wrong! *name1* and *name2* cannot be liar at the same time",
        "E": "Wrong! There is a case where *name1* is sincere"
      }}
```

This workflow is managed by a dedicated function that generates distinct instantiations of each question, randomises the order of the answer choices, and appends the corresponding feedback messages. The delivery of the quizzes to the user is handled by a separate function that employs the `inquirer`[2] package to present the questions sequentially. Through `inquirer`, users are able to navigate and select responses using the keyboard arrow keys. After each submission, the system immediately informs the user whether the selected answer is correct; in the case of an incorrect response, the user is prompted to try again and, if the feedback option has been enabled, is provided with a hint. The same question is repeated until the correct solution is chosen. In addition to recording the number of attempts, the program logs timestamps and question identifiers, storing them in a file that is exported at the end of the session.

10.5 Comparison with LLM

We also conducted an evaluation of ChatGPT's ability to solve the generated puzzles, with the aim of assessing how effectively a state-of-the-art language model handles this task.

We evaluated the model on a small set of puzzles from Mentor, and the outcomes varied: in some cases the solutions were correct, in others they were incorrect, and occasionally the answer was correct but the underlying reasoning was wrong. To better understand the outcome we decided to analyse the well-known Zebra Puzzle, considering three of the most widely used LLMs at present: ChatGPT, Gemini, and DeepSeek. As this puzzle is extensively documented in the literature, the models were able to solve it with relative ease, a result that can be attributed to their probabilistic nature and the likelihood of having encountered similar instances during training. To introduce a greater level of difficulty, we then designed a novel variant of the puzzle, formulated as follows:

1. *In the wonderful lagoon of Grado there are 4 houses of different colors, listed from west to east.*[3]
2. *In each house lives a family that speaks a different dialect.*
3. *Each family drinks one and only one beverage.*
4. *The Gradese does not live in the pink house.*
5. *The Istrian lives next to the Carnic.*
6. *The Gradese drinks spritz.*
7. *The Carnic does not drink water. Never.*
8. *Near the Gradese, people drink either water or beer.*
9. *Whoever drinks Ribolla does not live in a red or green house.*

[2] `https://pypi.org/project/inquirer/`

[3] The correct formulation is "casone" namely small buildings made of wood and hay where fishermen lived

10. The white house is, from left to right, between the green and the red one.
11. The person in the white house lives further west than the one who doesn't drink alcohol.
12. The Gradese and the Bisiaco are not neighbors.

Question: Who lives in the white house?

We encoded the puzzle in ASP and got that the only correct answer is that the Carnic lives in the white house.

Asking LLMs to solve it produced the following outputs:

- Gemini[4] concludes that the Gradese lives in the white house and argues incorrectly violating several conditions of the riddle;
- ChatGPT[5] produces a more detailed line of reasoning; however, its final answer (that it "The Istrian lives in the white house") is still incorrect. In particular, the model appears to misinterpret constraint 10, which is typically understood as requiring the white house to be positioned between the green and red houses, in strict left-to-right order and adjacent to both;
- DeepSeek[6] provides the correct final answer, but the solution reveals that constraint 10 is not fully respected; seems to stem from the model's difficulty in interpreting the phrase "from left to right" as denoting spatial orientation (west to east).

None of the models can be considered fully reliable, although some demonstrate stronger logical reasoning capabilities than others. In particular, DeepSeek R1 produced solutions that were almost both correct and complete; however, its reliability was strongly influenced by the clarity with which the problem was formulated.

10.6 Results

The experiment was conducted on first-year Computer Science students at the University of Udine during Programming Lab classes in March 2025. A total of 42 students volunteered to participate. The software was deployed on multiple laboratory workstations, and each student completed the test individually and anonymously. No time constraints were imposed, allowing the activity to resemble a training session rather than a formal assessment. All participants were presented with the same set of twelve question templates, instantiated with variable data and shown in random order. Students were divided into two groups: the first group (22 students) received quizzes with extended feedback, while the other group (20 students) was presented with only minimal feedback, *i.e.* a binary indication of correctness. At the end of the session, students completed an anonymous questionnaire in order to capture their perceptions of the difficulty of the questions and the usefulness of the feedback. The aim of the experiment

[4]premium version of December 2024
[5]-o1 premium version of December 2024
[6]R1 free version of February 2025

was to assess the effectiveness of feedback on incorrect answers and to collect qualitative insights from the users.

Even if the relatively small sample size prevents drawing statistically significant conclusions, several observations can be reported. Overall, the performance of the two groups was comparable, with a modest advantage observed in the extended feedback group. Students in the minimal feedback group typically required a larger number of attempts before arriving at the correct solution. Indeed, the duration of the activity varied between 10 and 20 minutes per student, but a slightly uneven distribution across the two groups emerged: the group receiving minimal feedback generally required more time to complete the test than the group receiving extended feedback. These preliminary findings suggest that providing explanatory feedback may support more efficient problem solving and could encourage deeper reflection on errors, thereby reinforcing the learning process.

In detail, the statistics for the number of correct answers on the first attempt and for the total number of failed attempts are reported in Table 10.2.

Condition	**Mean**	**Std. Dev.**	**Maximum**
Statistics for the number of attempts			
Extended feedback	5.72	2.49	11
Minimal feedback	6.10	2.51	10
Statistics for the total number of failed attempts			
Extended feedback	13.14	6.39	24
Minimal feedback	14.25	6.98	26

Table 10.2: Results for number of attempts and failed attempts under different feedback conditions

Analysis of the post-test questionnaires indicates that most students perceived the quizzes as moderately difficult. While the feedback mechanism was generally acknowledged, many respondents considered its usefulness to be limited. Given these results, it could be argued that feedback played only a marginal role. Nevertheless, feedback remains a crucial component of a learning tool of this nature: the findings suggest the need to improve both the design and the delivery of feedback so as to enhance its effectiveness and its perceived value among learners.

10.7 Future Works and Conclusions

With the growing popularity of large language models, such as ChatGPT, Gemini, and Deepseek, it is natural for students to view chatbots as potential tools to support test and exam preparation. However, current LLMs do not yet guarantee sufficient reliability, as their responses may lack consistency and correctness. In this context, a structured system such as the one described above offers a more robust and trustworthy alternative.

Developing a digital version of the Mentor raises several challenges, including the selection of an appropriate collection of questions, ensuring sufficient variety, guaranteeing that answers are consistent and unambiguous, and generating clear and effective feedback for learners. These aspects have been addressed in the present work, with en-

couraging results. Furthermore, the involvement of students in the experimental phase provided valuable suggestions for improvement. Among the most relevant proposals are: the development of a more intuitive and accessible graphical interface; the upgrade of feedback mechanisms to better highlight the underlying logical reasoning; and the integration of features tailored to students with special educational needs (BES).

Conclusions

In this work, we presented our research on combining symbolic reasoning and neural networks to achieve explainability in AI systems. The idea originated from the recognition that, while deep learning models achieve remarkable predictive performance across a variety of domains, they often lack transparency and interpretability. This limitation makes it difficult to adopt in critical fields, such as medicine and law, where decisions must be not only accurate but also justifiable to human experts. Therefore, the central aim of this work was to demonstrate that the integration of ILP with modern deep learning architectures can lead to models that not only perform well in terms of prediction accuracy but also provide human-understandable explanations of their outputs.

Summary of Research Project The work began with a study of the theoretical foundations of logic programming, answer set programming, and inductive logic programming, as well as an analysis of neural approaches to image classification and object detection, with particular emphasis on convolutional networks and YOLO. Building on these foundations, we proposed a hybrid pipeline in which neural networks are first employed to perform high-accuracy predictions, and the resulting knowledge is subsequently processed through ILP, in particular FastLAS and ILASP, to derive symbolic rules. These rules, represented in ASP, offer interpretable explanations of the system's decision-making process.

The methodology was evaluated across three distinct domains: weather forecasting, juridical reasoning, and veterinary medicine. Each case study provided different challenges in terms of data representation and domain complexity, yet together they demonstrated the generality of the proposed approach. Naturally, the research also revealed a number of limitations. In particular, the scalability of ILP-based approaches remains an open issue, especially when applied to high-dimensional data or large-scale problems. Furthermore, the quality of symbolic explanations strongly depends on the structure of the background knowledge and the expressiveness of the chosen representations.

Cross-case Analysis Despite the heterogeneity of the considered domains, a number of consistent patterns emerge from the experimental evaluation. Across all case studies, the proposed hybrid pipeline demonstrates that symbolic explanations can be extracted from neural models and/or data without substantially worsen predictive performance.

In the weather forecasting tasks, the learned rules capture spatio-temporal regularities that are well aligned with domain knowledge, thereby providing explanations that are not only faithful to the neural model's predictions but also meaningful to domain experts. In the juridical domain, the induced ASP programs reflect normative structures closely related to the underlying legal texts, highlighting the ability of ILP to encode exceptions, priorities, and non-monotonic reasoning which are features difficult to represent with purely statistical models. In the veterinary medicine case study, the

rules extracted from image-based predictions reveal biologically interpretable relationships between morphological features and classification outcomes, offering explanations that can support expert validation and trust.

Generalisability and Validity Threats While the experimental results are encouraging, several aspects must be considered when assessing their generalisability. First, the approach assumes the availability of meaningful symbolic predicates derived from the neural component or from domain knowledge. In domains where feature extraction is less structured or where intermediate concepts are difficult to formalise, the quality of the learned explanations may be reduced. This represents a construct validity threat, as the expressiveness of the explanation is partially dependent on the chosen symbolic representation.

Second, scalability remains a relevant limitation. Although systems such as FastLAS mitigate some of the computational bottlenecks of traditional ILP, the size of the hypothesis space can grow rapidly with the number of predicates and examples. This poses a potential external validity threat when considering large-scale, high-dimensional settings. Nevertheless, the experiments show that, for realistic problem sizes in applied domains, the approach remains practically viable.

Finally, the use of cross-validation and multiple datasets mitigates, but does not entirely remove, the risk of overfitting to specific experimental conditions. Future studies involving additional domains and alternative neural architectures would further strengthen the generality of the conclusions.

Alternative Approaches and Baselines Compared to post-hoc, model-agnostic explainability techniques such as LIME and SHAP, the proposed approach offers explanations that are global, structured, and grounded in a formal semantics. While LIME and SHAP are effective at highlighting local feature importance, they typically lack guarantees of logical consistency and struggle to express relational or non-monotonic dependencies. In contrast, ASP-based explanations explicitly encode such dependencies, enabling richer forms of reasoning and inspection.

When compared with end-to-end symbolic or neuro-symbolic approaches, the proposed pipeline adopts a modular design that preserves the strengths of deep learning in perception-heavy tasks while delegating explanation to a dedicated symbolic learner. This separation of concerns facilitates interpretability without requiring modifications to the neural architecture itself, making the approach compatible with state-of-the-art models such as YOLO.

From a learning perspective, the ILP-based explanations differ fundamentally from standard baselines such as decision trees or rule learners trained directly on raw data. Rather than approximating the input–output mapping, the learned rules aim to characterise the decision logic of an existing model, resulting in explanations that are faithful by construction. This distinction positions the contribution of this thesis within the work on faithful and verifiable XAI.

Advances of the Field Overall, the experimental findings advance the field of explainable AI in three main ways. First, they provide empirical evidence that ILP can be effectively integrated with modern deep learning systems to produce explanations

that are both accurate and semantically meaningful. Second, they demonstrate that non-monotonic logic programming is particularly well suited for explaining complex decision processes involving exceptions, priorities, and contextual dependencies. Third, they show that a single, unifying methodology can be applied across diverse domains, supporting the claim that the proposed framework is not merely application-specific but methodologically general.

Acronyms

AI	*Artificial Intelligence*
ANN	*Artificial Neural Network*
AP	*Average Precision*
ASP	*Answer Set Programming*
BB	*black-box*
BBSE	*Bull Breeding Soundness Evaluation*
BPTT	*Backpropagation Through Time*
CDCL	*Conflict-Driven Clause Learning*
CDILP	*Conflict-Driven Inductive Logic Programming*
CDPI	*Context-Dependent Partial Interpretation*
CERRA	*Copernicus European Regional ReAnalysis*
CNN	*Convolutional Neural Network*
ConvLSTM	*Convolutional LSTM*
DAG	*Directed Acyclic Graph*
DL	*Deep Learning*
DNF	*disjunctive normal form*
ECMWF	*European Centre for Medium-Range Weather Forecasts*
ELU	*Exponential Linear Unit*
EPS	*Ensemble Prediction System*
FastLAS	*Fast Learning of Answer Set Programs*
FiRE	*FIdelity vs. REadability*
FOL	*First Order Logic*
GDPR	*European General Data Protection Regulation*
GIoU	*Generalized Intersection over Union*
HCT	*Hough Circle Transform*
ICE	*Index for Complete quality Evaluation*

IFS	*Integrated Forecasting System*
ILASP	*Inductive Learning of Answer Set Programs*
ILP	*Inductive Logic Programming*
IoU	*Intersection over Union*
KR	*Knowledge Representation*
LAS	*Learning from Answer Set*
LFE	*Learning From Entailment*
LFI	*Learning From Interpretation*
LFS	*Learning From Satisfiability*
LIME	*Local Interpretable Model-agnostic Explanations*
LLM	*Large Language Model*
LP	*Logic Programming*
LSTM	*Long Short Term Memory*
mAP	*mean Average Precision*
NAF	*Negation As Failure*
NL	*Natural Language*
NLP	*Natural Language Processing*
NWP	*Numerical Weather Prediction*
OPL	*Observational Predicate Learning*
PSS	*Peirce Skill Score*
R-CNN	*Region-based CNN*
ReLU	*Rectified Linear Unit*
RNN	*Recurrent Neural Network*
s(CASP)	*solver for Constraints Answer Set Programs*
SELU	*Scaled Exponential Linear Unit*
SFT	*Society for Theriogenology*
SHAP	*SHapley Additive exPlanations*
SIRF	*Sistema Italiano Rilevamento Fulmini*
SKE	*Symbolic Knowledge Extraction*
SVM	*Support Vector Machine*
tobac	*Tracking and Object-Based Analysis of Clouds*
WCDPI	*Weighted Context-Dependent Partial Interpretation*
XAI	*Explainable Artificial Intelligence*

XAI-BSC	*XAI Bull Spermatozoa Classification*
xASP	*eXplainable Answer Set Programming*
YOLO	*You Only Look Once*

Bibliography

[1] Alan Mathison Turing. Computing machinery and intelligence. *Mind*, 49:433–460, 1950.

[2] Atul Rawal, James McCoy, Danda B. Rawat, Brian M. Sadler, and Robert St. Amant. Recent advances in trustworthy explainable artificial intelligence: Status, challenges, and perspectives. *IEEE Transactions on Artificial Intelligence*, 3(6):852–866, 2022.

[3] BBC. Amazon scrapped 'sexist AI' tool. `https://www.bbc.com/news/technology-45809919?utm_source=chatgpt.com`. Accessed: 10-10-2018.

[4] Luke Roberts, Harpreet Dhanoa, Sadie Lanes, and Jonathan Holdship. Machine learning for enhanced healthcare: an overview for operational and clinical leads. *British Journal of Healthcare Management*, 29(1):12–19, 2023.

[5] Stephen H. Muggleton. Inductive logic programming. *New generation computing*, 8(4):295–318, 1991.

[6] Katsumi Inoue, Andrei Doncescu, and Hidetomo Nabeshima. Completing causal networks by meta-level abduction. *Machine Learning*, 91:239–277, 2013.

[7] Domenico Corapi, Alessandra Russo, and Emil Lupu. Inductive logic programming in answer set programming. In Stephen H. Muggleton, Alireza Tamaddoni-Nezhad, and Francesca A. Lisi, editors, *Inductive Logic Programming - 21st International Conference, ILP 2011, Windsor Great Park, UK, July 31 - August 3, 2011, Revised Selected Papers*, volume 7207 of *Lecture Notes in Computer Science*, pages 91–97. Springer, 2011.

[8] Mark Law, Alessandra Russo, and Krysia Broda. Inductive learning of answer set programs. In Eduardo Fermé and João Leite, editors, *Logics in Artificial Intelligence*, pages 311–325, Cham, 2014. Springer International Publishing.

[9] Andrew Cropper and Rolf Morel. Learning programs by learning from failures. *Machine Learning*, 110(4):801–856, 2021.

[10] Tobias Kaminski, Thomas Eiter, and Katsumi Inoue. Exploiting answer set programming with external sources for meta-interpretive learning. *Theory and Practice of Logic Programming*, 18(3-4):571–588, 2018.

[11] Andrew Cropper and Sebastijan Dumančić. Learning large logic programs by going beyond entailment. *arXiv preprint arXiv:2004.09855*, 2020.

[12] Luc De Raedt, Kristian Kersting, Sriraam Natarajan, and David Poole. Statistical relational artificial intelligence: Logic, probability, and computation. *Synthesis lectures on artificial intelligence and machine learning*, 10(2):1–189, 2016.

[13] Luc De Raedt and Kristian Kersting. Probabilistic inductive logic programming. In *Probabilistic inductive logic programming: theory and applications*, pages 1–27. Springer, 2008.

[14] Mark Law, Alessandra Russo, and Krysia Broda. The ILASP system for inductive learning of answer set programs. *arXiv preprint arXiv:2005.00904*, 2020.

[15] Mark Law, Alessandra Russo, Elisa Bertino, Krysia Broda, and Jorge Lobo. FastLAS: Scalable inductive logic programming incorporating domain-specific optimisation criteria. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(03):2877–2885, Apr. 2020.

[16] Talissa Dreossi. Exploring ILASP through logic puzzles modelling. In Agostino Dovier and Andrea Formisano, editors, *Proceedings of the 38th Italian Conference on Computational Logic, Udine, Italy, June 21-23, 2023*, volume 3428 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023.

[17] Kexin Gu Baugh, Nuri Cingillioglu, and Alessandra Russo. Neuro-symbolic rule learning in real-world classification tasks. In Andreas Martin, Hans-Georg Fill, Aurona Gerber, Knut Hinkelmann, Doug Lenat, Reinhard Stolle, and Frank van Harmelen, editors, *Proc. of AAAI-MAKE 2023*, volume 3433 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023.

[18] Daniel Cunnington, Flaviu Cipcigan, Rodrigo Neumann Barros Ferreira, and Jonathan Booth. Symbolic learning for material discovery. *arXiv preprint arXiv:2312.11487*, 2023.

[19] Arthur Drozdov, Mark Law, Jorge Lobo, Alessandra Russo, and Mercion Wilathgamuwage Don. Online symbolic learning of policies for explainable security. In *2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, pages 269–278, 2021.

[20] Seraphin Calo, Irene Manotas, Geeth de Mel, Daniel Cunnington, Mark Law, Dinesh Verma, Alessandra Russo, and Elisa Bertino. *AGENP: An ASGrammar-based GENerative Policy Framework*, pages 3–20. Springer International Publishing, Cham, 2019.

[21] Daniel Cunnington, Mark Law, Jorge Lobo, and Alessandra Russo. FFNSL: Feed-forward neural-symbolic learner. *Machine Learning*, 112(2):515–569, 2023.

[22] European Parliament and Council of the European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council.

[23] Nick Bostrom and Eliezer Yudkowsky. The ethics of artificial intelligence. In *Artificial intelligence safety and security*, pages 57–69. Chapman and Hall/CRC, 2018.

[24] Daniel S Weld and Gagan Bansal. The challenge of crafting intelligible intelligence. *Communications of the ACM*, 62(6):70–79, 2019.

[25] Arun Das and Paul Rad. Opportunities and challenges in explainable artificial intelligence (XAI): A survey. *arXiv preprint arXiv:2006.11371*, 2020.

[26] Robert Challen, Joshua Denny, Martin Pitt, Luke Gompels, Tom Edwards, and Krasimira Tsaneva-Atanasova. Artificial intelligence, bias and clinical safety. *BMJ quality & safety*, 28(3):231–237, 2019.

[27] Mario Alviano, Ly Ly T. Trieu, Tran Cao Son, and Marcello Balduccini. Advancements in xASP, an XAI System for Answer Set Programming. In Agostino Dovier and Andrea Formisano, editors, *Proceedings of the 38th Italian Conference on Computational Logic, Udine, Italy, June 21-23, 2023*, volume 3428 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023.

[28] Pedro Cabalar, Jorge Fandinno, and Michael Fink. Causal graph justifications of logic programs. *Theory and Practice of Logic Programming*, 14(4-5):603–618, 2014.

[29] Thomas G. Dietterich. Overfitting and undercomputing in machine learning. *ACM Comput. Surv.*, 27:326–327, 1995.

[30] Andrew Cropper, Sebastijan Dumančić, Richard Evans, and Stephen H Muggleton. Inductive logic programming at 30. *Machine Learning*, pages 1–26, 2022.

[31] Maarten H. van Emden and Robert A. Kowalski. The semantics of predicate logic as a programming language. *J. ACM*, 23(4):733–742, 1976.

[32] Michael Gelfond and Yulia Kahl. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*. Cambridge University Press, March 2014.

[33] Victor W. Marek and Miroslaw Truszczynski. Stable models and an alternative logic programming paradigm. In Krzysztof R. Apt, Victor W. Marek, Mirek Truszczynski, and David Scott Warren, editors, *The Logic Programming Paradigm - A 25-Year Perspective*, Artificial Intelligence, pages 375–398. Springer, 1999.

[34] Stefano Ceri, Georg Gottlob, Letizia Tanca, et al. What you always wanted to know about Datalog (and never dared to ask). *IEEE transactions on knowledge and data engineering*, 1(1):146–166, 1989.

[35] Calvin Jongsma. *Introduction to Discrete Mathematics via Logic and Proof (Chapter 2: First-Order Logic)*. Springer International Publishing, Cham, 2019.

[36] John Wylie Lloyd. *Foundations of Logic Programming*. Springer Berlin Heidelberg, 1987.

[37] Keith L. Clark. *Negation as Failure*, pages 293–322. Springer US, Boston, MA, 1978.

[38] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer set solving in practice*. Springer Nature, 2022.

[39] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Proc. of ICLP/SLP*, pages 1070–1080. MIT Press, 1988.

[40] Luc De Raedt. Logical settings for concept-learning. *Artificial Intelligence*, 95(1):187–201, 1997.

[41] Mark Law, Alessandra Russo, and Krysia Broda. Inductive learning of answer set programs from noisy examples. *arXiv preprint arXiv:1808.08441*, 2018.

[42] Mark Law. *Inductive learning of answer set programs*. PhD thesis, Imperial College London, UK, 2018.

[43] Mark Law. Conflict-driven inductive logic programming. *Theory and Practice of Logic Programming*, 23(2):387–414, 2023.

[44] Mark Law, Alessandra Russo, and Krysia Broda. Iterative learning of answer set programs from context dependent examples. *Theory and Practice of Logic Programming*, 16(5-6):834–848, 2016.

[45] Mark Law, Alessandra Russo, Krysia Broda, and Elisa Bertino. Scalable non-observational predicate learning in ASP. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 1936–1943. ijcai.org, 2021.

[46] Giulia Vilone and Luca Longo. Notions of explainability and evaluation approaches for explainable artificial intelligence. *Information Fusion*, 76:89–106, 2021.

[47] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

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

[49] Timo Speith. A review of taxonomies of explainable artificial intelligence (XAI) methods. In *2022 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '22, page 2239–2250. ACM, June 2022.

[50] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Computing Surveys*, 51(5):1–42, August 2018.

[51] Mario Alviano, Ly Ly T. Trieu, Tran Cao Son, and Marcello Balduccini. The XAI system for answer set programming xASP2. *J. Log. Comput.*, 34(8):1500–1525, 2024.

[52] Leilani H. Gilpin, David Bau, Ben Z. Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 80–89, 2018.

[53] Enrico Pontelli, Tran Cao Son, and Omar Elkhatib. Justifications for logic programs under answer set semantics. *Theory and Practice of Logic Programming*, 9(1):1–56, 2009.

[54] Pedro Cabalar, Jorge Fandinno, and Brais Muñiz. A system for explainable answer set programming. *arXiv preprint arXiv:2009.10242*, 2020.

[55] Fang Li, Huaduo Wang, Kinjal Basu, Elmer Salazar, and Gopal Gupta. DiscASP: A graph-based ASP system for finding relevant consistent concepts with applications to conversational socialbots. *arXiv preprint arXiv:2109.08297*, 2021.

[56] Ly Ly Trieu, Tran Cao Son, and Marcello Balduccini. XASP: An explanation generation system for answer set programming. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 363–369. Springer, 2022.

[57] Ly Ly Trieu, Tran Cao Son, and Marcello Balduccini. exp (aspc): Explaining asp programs with choice atoms and constraint rules. *arXiv preprint arXiv:2109.08292*, 2021.

[58] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the predictions of any classifier. *arXiv preprint arXiv:1602.04938*, 2016.

[59] Scott Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *arXiv preprint: arXiv:1705.07874*, 2017.

[60] Shier Nee Saw, Yet Yen Yan, and Kwan Hoong Ng. Current status and future directions of explainable artificial intelligence in medical imaging. *European Journal of Radiology*, 183:111884, 2025.

[61] Ahmed M Salih, Zahra Raisi-Estabragh, Ilaria Boscolo Galazzo, Petia Radeva, Steffen E Petersen, Karim Lekadir, and Gloria Menegaz. A perspective on explainable artificial intelligence methods: SHAP and LIME. *Advanced Intelligent Systems*, 7(1):2400304, 2025.

[62] Narinder Singh Punn and Sonali Agarwal. Automated diagnosis of COVID-19 with limited posteroanterior chest x-ray images using fine-tuned deep neural networks. *Applied Intelligence*, 51(5):2689–2702, 2021.

[63] Pavan Rajkumar Magesh, Richard Delwin Myloth, and Rijo Jackson Tom. An explainable machine learning model for early detection of parkinson's disease using LIME on DaTSCAN imagery. *Computers in biology and medicine*, 126:104041, 2020.

[64] Asif Rahman, Maqsood Hayat, Nadeem Iqbal, Fawaz Khaled Alarfaj, Salem Alkhalaf, and Fahad Alturise. Enhanced MRI brain tumor detection using deep learning in conjunction with explainable AI SHAP based diverse and multi feature analysis. *Scientific Reports*, 15(1):29411, 2025.

[65] Md Nahiduzzaman, Lway Faisal Abdulrazak, Mohamed Arselene Ayari, Amith Khandakar, and SM Riazul Islam. A novel framework for lung cancer classification using lightweight convolutional neural networks and ridge extreme learning machine model with SHapley Additive exPlanations (SHAP). *Expert Systems with Applications*, 248:123392, 2024.

[66] Andrea Agiollo and Andrea Omicini. Measuring trustworthiness in neuro-symbolic integration. In *Annals of Computer Science and Information Systems*, volume 35, pages 1–10. IEEE, September 2023.

[67] Simin Chen, Soroush Bateni, Sampath Grandhi, Xiaodi Li, Cong Liu, and Wei Yang. DENAS: automated rule generation by knowledge extraction from neural networks. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, New York, NY, USA, November 2020. ACM.

[68] Maxwell Nye, Michael Tessler, Josh Tenenbaum, and Brenden M Lake. Improving coherence and consistency in neural sequence models with dual-system, neuro-symbolic reasoning. *Advances in Neural Information Processing Systems*, 34:25192–25204, 2021.

[69] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Matroids, matchings, and fairness. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18 April 2019, Naha, Okinawa, Japan*, volume 89 of *Proceedings of Machine Learning Research*, pages 2212–2220. PMLR, 2019.

[70] Sorelle A. Friedler, Carlos Scheidegger, and Suresh Venkatasubramanian. The (im)possibility of fairness: different value systems require different mechanisms for fair decision making. *Communications of the ACM*, 64(4):136–143, March 2021.

[71] Federico Sabbatini. *Exploiting Explainable Artificial Intelligence for Space Weather Investigations on board LISA and Future Space Interferometers for Gravitational Wave Detection*. Phd thesis, Università degli Studi "Carlo Bo" di Urbino, 2025.

[72] Federico Sabbatini and Roberta Calegari. On the evaluation of the symbolic knowledge extracted from black boxes. *AI and Ethics*, 4(1):65–74, January 2024.

[73] Federico Sabbatini and Roberta Calegari. Symbolic knowledge-extraction evaluation metrics: The FiRe score. In *ECAI 2023*, pages 2033–2040. IOS Press, 2023.

[74] Federico Sabbatini and Roberta Calegari. *ICE: An Evaluation Metric to Assess Symbolic Knowledge Quality*, page 241–256. Springer Nature Switzerland, 2025.

[75] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[76] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[77] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[78] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, real-time object detection, 2016.

[79] Lewis F. Richardson. *Weather Prediction by Numerical Process*. Cambridge Mathematical Library. Cambridge University Press, 1922.

[80] Juanzhen Sun, Ming Xue, James W. Wilson, Isztar Zawadzki, Sue P. Ballard, Jeanette Onvlee-Hooimeyer, Paul Joe, Dale M. Barker, Ping-Wah Li, Brian Golding, Mei Xu, and James Pinto. Use of NWP for nowcasting convective precipitation: Recent progress and challenges. *Bulletin of the American Meteorological Society*, 95(3):409 – 426, 2014.

[81] Eugenia Kalnay. *Atmospheric Modeling, Data Assimilation and Predictability*. Cambridge University Press, 2002.

[82] Stephan Rasp, Michael S Pritchard, and Pierre Gentine. Deep learning to represent subgrid processes in climate models. *Proceedings of the national academy of sciences*, 115(39):9684–9689, 2018.

[83] Amy McGovern, Kimberly L Elmore, David John Gagne, Sue Ellen Haupt, Christopher D Karstens, Ryan Lagerquist, Travis Smith, and John K Williams. Using artificial intelligence to improve real-time decision-making for high-impact weather. *Bulletin of the American Meteorological Society*, 98(10):2073–2090, 2017.

[84] Yunxiang Liu and Shixuan Cai. Advancements in weather forecasting: A review of satellite imagery and AI integration. In *2024 9th International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, volume 9, pages 340–344, 2024.

[85] Kanghui Zhou, Yongguang Zheng, Bo Li, Wansheng Dong, and Xiaoling Zhang. Forecasting different types of convective weather: A deep learning approach. *Journal of Meteorological Research*, 33:797–809, 2019.

[86] Fengyuan Zhang. Weather forecasting and analysis with lstm based on deep learning. In *Proceedings of the 2024 2nd International Conference on Image, Algorithms and Artificial Intelligence*, volume 2, page 165. Springer Nature, 2024.

[87] Jonathan A Weyn, Dale R Durran, and Rich Caruana. Can machines learn to predict weather? Using deep learning to predict gridded 500-hPa geopotential height from historical weather data. *J. of Advances in Modeling Earth Systems*, 11(8):2680–2693, 2019.

[88] Jonathan A Weyn, Dale R Durran, and Rich Caruana. Improving data-driven global weather prediction using deep convolutional neural networks on a cubed sphere. *J. of Advances in Modeling Earth Systems*, 12(9):e2020MS002109, 2020.

[89] Lizao Li, Robert Carver, Ignacio Lopez-Gomez, Fei Sha, and John Anderson. Generative emulation of weather forecast ensembles with diffusion models. *Science Advances*, 10(13), 2024.

[90] M. Heikenfeld, P. J. Marinescu, M. Christensen, D. Watson-Parris, F. Senf, S. C. van den Heever, and P. Stier. tobac 1.2: towards a flexible framework for tracking and analysis of clouds in diverse datasets. *Geoscientific Model Development*, 12(11):4551–4570, 2019.

[91] G. A. Sokolowsky, S. W. Freeman, W. K. Jones, J. Kukulies, F. Senf, P. J. Marinescu, M. Heikenfeld, K. N. Brunner, E. C. Bruning, S. M. Collis, R. C. Jackson, G. R. Leung, N. Pfeifer, B. A. Raut, S. M. Saleeby, P. Stier, and S. C. van den Heever. *tobac* v1.5: introducing fast 3d tracking, splits and mergers, and other enhancements for identifying and analysing meteorological phenomena. *Geoscientific Model Development*, 17(13):5309–5330, 2024.

[92] J. Bukowski and S. C. van den Heever. Direct radiative effects in haboobs. *Journal of Geophysical Research: Atmospheres*, 126(21), October 2021.

[93] Julia Kukulies, Deliang Chen, and Julia Curio. The role of mesoscale convective systems in precipitation in the tibetan plateau region. *Journal of Geophysical Research: Atmospheres*, 126(23), December 2021.

[94] Yang Li, Yubao Liu, Yun Chen, Baojun Chen, Xin Zhang, Weisheng Wang, Zhuozhi Shu, and Zhaoyang Huo. Characteristics of deep convective systems and initiation during warm seasons over china and its vicinity. *Remote Sensing*, 13(21), 2021.

[95] G. R. Leung, S. M. Saleeby, G. A. Sokolowsky, S. W. Freeman, and S. C. van den Heever. Aerosol–cloud impacts on aerosol detrainment and rainout in shallow maritime tropical clouds. *Atmospheric Chemistry and Physics*, 23(9):5263–5278, 2023.

[96] Zachary M Labe, Nathaniel Johnson, and Thomas L Delworth. Changes in United States summer temperatures revealed by explainable neural networks. *Authorea Preprints*, 2023.

[97] Benjamin A Toms, Elizabeth A Barnes, and James W Hurrell. Assessing decadal predictability in an earth-system model using explainable neural networks. *Geophysical Research Letters*, 48(12):e2021GL093842, 2021.

[98] Talissa Dreossi, Agostino Dovier, Andrea Formisano, Mark Law, Agostino Manzato, Alessandra Russo, and Matthew Tait. Towards explainable weather forecasting through FastLAS. In Carmine Dodaro, Gopal Gupta, and Maria Vanina Martinez, editors, *Logic Programming and Nonmonotonic Reasoning - 17th International Conference, LPNMR 2024, Dallas, TX, USA, October 11-14, 2024,*

Proceedings, volume 15245 of *Lecture Notes in Computer Science*, pages 262–275. Springer, 2024.

[99] ARPA FVG. *Riepilogo anno 2022*, chapter 1. SOC OSMER e GRN - Osservatorio Meteorologico Regionale, 2023.

[100] Agostino Manzato. A note on the maximum Peirce skill score. *Weather and Forecasting*, 22(5):1148–1154, 2007.

[101] Davide Soldà. *Implementazione del software Atmoswing per la previsione dei temporali in FVG tramite metodo dell'analogo*. Bachelor's thesis, University of Udine, 2018–2019.

[102] Luca Foschiani. *Deep Learning per la previsione di fenomeni temporaleschi in Friuli Venezia Giulia*. Bachelor's thesis, University of Udine, 2018–2019.

[103] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai kin Wong, and Wang chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting, 2015.

[104] Hla Hart. *The Concept of Law*. Oxford University Press UK, Oxford, United Kingdom, 1961.

[105] Agostino Dovier, Talissa Dreossi, Andrea Formisano, and Benedetta Strizzolo. XAI-LAW a logic programming tool for modeling, explaining, and learning legal decisions, September, 2025. Presented at ICLP 2025, International Conference of Logic Programming, University of Calabria, Italy.

[106] Agostino Dovier, Talissa Dreossi, and Andrea Formisano. XAI-LAW towards a logic programming tool for taking and explaining legal decisions. In Emanuele De Angelis and Maurizio Proietti, editors, *Proceedings of the 39th Italian Conference on Computational Logic, Rome, Italy, June 26-28, 2024*, volume 3733 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2024.

[107] Manuele Dozzi, Talissa Dreossi, Federico Costantini, Agostino Dovier, and Andrea Formisano. Semi-automatic knowledge representation and reasoning on vagueness crime concepts, 2023. ALP2023, JURIX 2023 workshop on AI, law and philosophy, Maastricht, Netherlands.

[108] Manuele Dozzi, Talissa Dreossi, Luca Baron, Federico Costantini, Agostino Dovier, and Andrea Formisano. Semi-automatic knowledge representation and reasoning on vague crime concepts, 2024. Workshop DigForASP, colocated with 15th European Symposium on Computational Intelligence and Mathematics. Krakow (Poland). May 12th-15th, 2024. Forthcoming in Computational Intelligence and Mathematics for Tackling Complex Problems.

[109] Layman E. Allen. Symbolic logic: A razor-edged tool for drafting and interpreting legal documents. *The Yale Law Journal*, 66(6):933–879, 1957.

[110] Layman E. Allen and Charles S. Saxon. Some problems in designing expert systems to aid legal reasoning. In *Proceedings of the First International Conference on Artificial Intelligence and Law, ICAIL '87, Boston, MA, USA, May 27-29, 1987*, pages 94–103. ACM, 1987.

[111] Robert A. Kowalski, Jacinto A. Dávila, Galileo Sartor, and Miguel Calejo. Logical english for law and education. In David Scott Warren, Verónica Dahl, Thomas Eiter, Manuel V. Hermenegildo, Robert A. Kowalski, and Francesca Rossi, editors, *Prolog: The Next 50 Years*, volume 13900 of *Lecture Notes in Computer Science*, pages 287–299. Springer, 2023.

[112] Robert A. Kowalski and Marek J. Sergot. Computer representation of the law. In Aravind K. Joshi, editor, *Proceedings of the 9th International Joint Conference on Artificial Intelligence. Los Angeles, CA, USA, August 1985*, pages 1269–1270. Morgan Kaufmann, 1985.

[113] Forouzan Golshani. Automated construction of legal arguments. *Int. J. Intell. Syst.*, 6(6):673–685, 1991.

[114] Galileo Sartor, Jacinto A. Dávila, Marco Billi, Giuseppe Contissa, Giuseppe Pisano, and Robert A. Kowalski. Integration of logical English and s(CASP). In *Proceedings of the International Conference on Logic Programming 2022 Workshops co-located with the 38th International Conference on Logic Programming (ICLP) 2022, Haifa, Israel, July 31st - August 1st, 2022*, volume 3193 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2022.

[115] D. Ludovica. *Manuale di diritto penale: parte speciale II*. Giuffre Francis Lefebvre, 2022.

[116] Enzo Musco Giovanni Fiandaca. *Diritto penale. Parte speciale. 2.2, I delitti contro il patrimonio*. Zanichelli, Bologna, 8 edition, 2023.

[117] Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2010.

[118] Kevin D. Ashley. *Artificial Intelligence and Legal Analytics: New Tools for Law Practice in the Digital Age*. Cambridge University Press, Cambridge, 2017.

[119] Agostino Dovier, Andrea Formisano, Enrico Pontelli, and Flavio Vella. A GPU implementation of the ASP computation. In Marco Gavanelli and John H. Reppy, editors, *Proc. of PADL 2016*, volume 9585 of *LNCS*, pages 30–47. Springer, 2016.

[120] Agostino Dovier, Andrea Formisano, and Enrico Pontelli. Parallel answer set programming. In Youssef Hamadi and Lakhdar Sais, editors, *Handbook of Parallel Constraint Reasoning*, pages 237–282. Springer, 2018.

[121] Agostino Dovier, Andrea Formisano, and Flavio Vella. GPU-based parallelism for ASP-solving. In Petra Hofstedt, Salvador Abreu, Ulrich John, Herbert Kuchen, and Dietmar Seipel, editors, *Declarative Programming and Knowledge Management*, volume 12057 of *LNCS*, pages 3–23. Springer, 2019.

[122] Agostino Dovier, Andrea Formisano, Gopal Gupta, Manuel V. Hermenegildo, Enrico Pontelli, and Ricardo Rocha. Parallel logic programming: A sequel. *Theory Pract. Log. Program.*, 22(6):905–973, 2022.

[123] Henry Prakken and Giovanni Sartor. Law and logic: A review from an argumentation perspective. *Artif. Intell.*, 227:214–245, 2015.

[124] Susy Urli, Francesca Corte Pause, Talissa Dreossi, Martina Crociati, and Giuseppe Stradaioli. Evaluation of an artificial intelligence system for bull sperm morphology evaluation. *Theriogenology*, 245:117504, 2025.

[125] Talissa Dreossi, Agostino Dovier, Susy Urli, Francesca Corte Pause, and Martina Crociati. Explainable AI for sperm morphology: Integrating YOLO with FastLAS. In Dario Guidotti, Laura Pandolfo, and Luca Pulina, editors, *Proceedings of the 40th Italian Conference on Computational Logic, Alghero, Italy, June 25-27, 2025*, volume 4003 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2025.

[126] Talissa Dreossi, Agostino Dovier, Susy Urli, Francesca Corte Pause, and Martina Crociati. Explainable AI for sperm morphology: Integrating YOLO with FastLAS, September, 2025. Presented at ICLP 2025, International Conference of Logic Programming, University of Calabria, Italy.

[127] Mohammad Shehab, Laith Abualigah, Qusai Shambour, Muhannad A. Abu-Hashem, Mohd Khaled Yousef Shambour, Ahmed Izzat Alsalibi, and Amir H. Gandomi. Machine learning in medical applications: A review of state-of-the-art methods. *Computers in Biology and Medicine*, 145:105458, June 2022.

[128] Ryan B. Appleby and Parminder S. Basran. Artificial intelligence in veterinary medicine. *Journal of the American Veterinary Medical Association*, 260(8):819–824, May 2022.

[129] Anders Krogh Mortensen, Pavel Lisouski, and Peter Ahrendt. Weight prediction of broiler chickens using 3D computer vision. *Computers and Electronics in Agriculture*, 123:319–326, April 2016.

[130] Francisco A. García-Vázquez. Artificial intelligence and porcine breeding. *Animal Reproduction Science*, 269:107538, October 2024.

[131] T. V. Raja, A. P. Ruhil, and R. S. Gandhi. Comparison of connectionist and multiple regression approaches for prediction of body weight of goats. *Neural Computing and Applications*, 21(1):119–124, May 2011.

[132] Oleksandr Bezsonov, Oleh Lebediev, Valentyn Lebediev, Yuriy Megel, Dmytro Prochukhan, and Oleg Rudenko. Breed recognition and estimation of live weight of cattle based on methods of machine learning and computer vision. *Eastern-European Journal of Enterprise Technologies*, 6(9 (114)):64–74, December 2021.

[133] Hend Radwan, Hadeel Qaliouby, and Eman Elfadl. Classification and prediction of milk yield level for Holstein Friesian cattle using parametric and non-parametric statistical classification models. *Journal of Advanced Veterinary and Animal Research*, 7(3):429, 2020.

[134] Ciara O'Meara, Emilie Henrotte, Kasia Kupisiewicz, Catherine Latour, Marleen Broekhuijse, Agnes Camus, Lucie Gavin-Plagne, and Eli Sellem. The effect of adjusting settings within a computer-assisted sperm analysis (CASA) system on bovine sperm motility and morphology results. *Animal Reproduction*, 19, 02 2022.

[135] David Gil, Jose Luis Girela, Joaquin De Juan, M. Jose Gomez-Torres, and Magnus Johnsson. Predicting seminal quality with artificial intelligence methods. *Expert Systems with Applications*, 39(16):12564–12573, 2012.

[136] Manesh Kumar Panner Selvam, Ajaya Kumar Moharana, Saradha Baskaran, Renata Finelli, Matthew C Hudnall, and Suresh C Sikka. Current updates on involvement of artificial intelligence and machine learning in semen analysis. *Medicina*, 60(2):279, 2024.

[137] Anna Badura, Urszula Marzec-Wróblewska, Piotr Kamiński, Paweł Łakota, Grzegorz Ludwikowski, Marek Szymański, Karolina Wasilow, Andżelika Lorenc, and Adam Buciński. Prediction of semen quality using artificial neural network. *Journal of Applied Biomedicine*, 17(3), 2019.

[138] Mikhail E. Kandel, Marcello Rubessa, Yuchen R. He, Sierra Schreiber, Sasha Meyers, Luciana Matter Naves, Molly K. Sermersheim, G. Scott Sell, Michael J. Szewczyk, Nahil Sobh, Matthew B. Wheeler, and Gabriel Popescu. Reproductive outcomes predicted by phase imaging with computational specificity of spermatozoon ultrastructure. *Proceedings of the National Academy of Sciences*, 117(31):18302–18309, July 2020.

[139] Hanson Rachel, Reddick Sadie, Thuerauf Sabrina, Webb Katie, Kasimanickam Vanmathy, and Kasimanickam Ramanathan. Comparison of bull sperm morphology evaluation methods under field conditions. *Clinical Theriogenology*, 15, Aug. 2023.

[140] Francesca Corte Pause, Martina Crociati, Susy Urli, Maurizio Monaci, Lorenzo Degano, and Giuseppe Stradaioli. Environmental factors affecting the reproductive efficiency of italian simmental young bulls. *Animals*, 12(18), 2022.

[141] Edda Dal Santo, Agostino Dovier, and Talissa Dreossi. TOLC-ASP: a tool for training students for admission tests. In Dario Guidotti, Laura Pandolfo, and Luca Pulina, editors, *Proceedings of the 40th Italian Conference on Computational Logic, Alghero, Italy, June 25-27, 2025*, volume 4003 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2025.

[142] Agostino Dovier, Andrea Formisano, and Enrico Pontelli. An empirical study of constraint logic programming and answer set programming solutions of combinatorial problems. *J. Exp. Theor. Artif. Intell.*, 21(2):79–121, 2009.

[143] Lorenzo Cian, Talissa Dreossi, and Agostino Dovier. Modeling and solving the rush hour puzzle. In Roberta Calegari, Giovanni Ciatto, and Andrea Omicini, editors, *Proceedings of the 37th Italian Conference on Computational Logic, Bologna, Italy, June 29 - July 1, 2022*, volume 3204 of *CEUR Workshop Proceedings*, pages 294–306. CEUR-WS.org, 2022.

[144] Nicola Rizzo and Agostino Dovier. 3coSoKu and its declarative modeling. *J. Log. Comput.*, 32(2):307–330, 2022.

[145] Agostino Dovier, Paolo Benoli, Maria Concetta Brocato, Luciano Dereani, and Federica Tabacco. Reasoning in High Schools: Do it with ASP! In Camillo Fiorentini and Alberto Momigliano, editors, *Proceedings of the 31st Italian Conference on Computational Logic, Milano, Italy, June 20-22, 2016*, volume 1645 of *CEUR Workshop Proceedings*, pages 205–213. CEUR-WS.org, 2016.

[146] Robert Kowalski, Jacinto Dávila, Galileo Sartor, and Miguel Calejo. Logical english for law and education. In David S. Warren, Veronica Dahl, Thomas Eiter, Manuel V. Hermenegildo, Robert Kowalski, and Francesca Rossi, editors, *Prolog: The Next 50 Years*, volume 13900, pages 287–299. Springer Nature Switzerland, Cham, 2023.

[147] Manuel V. Hermenegildo, Jose F. Morales, and Pedro Lopez-Garcia. Some thoughts on how to teach prolog. In David S. Warren, Veronica Dahl, Thomas Eiter, Manuel V. Hermenegildo, Robert Kowalski, and Francesca Rossi, editors, *Prolog: The Next 50 Years*, volume 13900, pages 107–123. Springer Nature Switzerland, Cham, 2023.

[148] Laura A. Cecchi, Jorge P. Rodríguez, and Veronica Dahl. Logic programming at elementary school: Why, what and how should we teach logic programming to children? In David S. Warren, Veronica Dahl, Thomas Eiter, Manuel V. Hermenegildo, Robert Kowalski, and Francesca Rossi, editors, *Prolog: The Next 50 Years*, volume 13900, pages 131–143. Springer Nature Switzerland, Cham, 2023.

[149] Giorgio Filippi and Vincenzo Falco. I risultati TOLC 2023. https://www.cisiaonline.it/sites/default/files/Divulgazione/2024/Volume-Risultati_TOLC_2023-CISIA.pdf, 2024.

[150] Luisella Caire and Paola Suria Arnaldi. *Mentor di Logica vol.1.* Cisia, 2018.

[151] Riccardo Orlando, Luca Moroni, Pere-Lluís Huguet Cabot, Simone Conia, Edoardo Barba, Sergio Orlandini, Giuseppe Fiameni, and Roberto Navigli. Minerva LLMs: The first family of large language models trained from scratch on italian data. In Felice Dell'Orletta, Alessandro Lenci, Simonetta Montemagni, and Rachele Sprugnoli, editors, *Proceedings of the Tenth Italian Conference on Computational Linguistics (CLiC-it 2024), Pisa, Italy, December 4-6, 2024*, volume 3878 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2024. https://nlp.uniroma1.it/minerva/.