Proceedings of the 23rd International Conference on Principles of Knowledge Representation and Reasoning
KR meets Machine Learning and Explanation

# Logic of Hypotheses: from Zero to Full Knowledge in Neurosymbolic Integration

**Davide Bizzaro**[1,2] , **Alessandro Daniele**[2,3]
[1]University of Padua, Padova, Italy
[2]Fondazione Bruno Kessler, Trento, Italy
[3]University of Bozen-Bolzano, Bozen, Italy
davide.bizzaro@phd.unipd.it, alessandro.daniele@unibz.it

**Abstract**

Neurosymbolic integration (NeSy) blends neural-network learning with symbolic reasoning. The field can be split between methods injecting hand-crafted rules into neural models, and methods inducing symbolic rules from data. We introduce *Logic of Hypotheses* (LoH), a novel language that unifies these strands, enabling the flexible integration of data-driven rule learning with symbolic priors and expert knowledge. LoH extends propositional logic syntax with a *choice operator*, which has learnable parameters and selects a subformula from a pool of options. Using fuzzy logic, formulas in LoH can be directly compiled into a differentiable computational graph, so the optimal choices can be learned via backpropagation. This framework subsumes some existing NeSy models, while adding the possibility of arbitrary degrees of knowledge specification. Moreover, the use of Gödel fuzzy logic and the recently developed Gödel trick yields models that can be discretized to hard Boolean-valued functions without any loss in performance. We provide experimental analysis on such models, showing strong results on tabular data and on two NeSy tasks with a perceptual component.

## 1 Introduction

Neurosymbolic integration (NeSy) tries to combine the symbolic and sub-symbolic paradigms. The aim is to retain the clarity and deductive power of logic while leveraging the learning capabilities of neural networks (Besold et al. 2021; Marra et al. 2024). In many NeSy approaches, such as DeepProbLog (Manhaeve et al. 2018) and LTN (Badreddine et al. 2022), domain experts provide prior knowledge in the form of logic formulas, which the neural model uses as a bias or as constraints. While this strategy has proven effective, it presupposes that high-quality rules are readily available. On the other hand, other NeSy methods have approached the learning of symbolic knowledge from data in the fields of rule mining (Qiao, Wang, and Lin 2021; Katzir, Elidan, and El-Yaniv 2020; Wang et al. 2020) and inductive logic programming (Evans and Grefenstette 2018; Rocktäschel and Riedel 2017). Further, some advanced methods can simultaneously learn symbolic rules and perception-to-symbol mappings (Wang et al. 2019; Daniele et al. 2022; Barbiero et al. 2023), thereby grounding symbols to raw data while simultaneously discovering the logical structure.

These research threads occupy opposite ends of a spectrum: knowledge-*injection* versus rule-*induction*. However, they typically lack the flexibility to handle intermediate scenarios in which a *partial logical structure* is supplied and the missing parts must still be learned. Such situations arise, for example, when existing prior knowledge must be revised or completed, or when learned rules are required to respect specific syntactic templates (e.g., CNF, DNF, Horn clauses, fixed-length clauses) for which the model was never programmed. Addressing this flexible middle ground remains an open challenge for NeSy methods.

In this paper, we propose the *Logic of Hypotheses* (LoH), a new logical formalism introducing the *choice operator*, which learns to select a subformula from a set of candidates. Such language can be used to produce neural networks with varying degree of symbolic bias, ranging from complete prior knowledge to none. In this way, LoH offers a single, principled learning framework that can adapt to the amount and form of prior knowledge available. Our main contributions are:

- **A novel language (LoH)** that extends propositional logic syntax with a *choice operator*, making it possible to leave parts of a formula underspecified. This allows to represent a hypothesis space $\mathcal{H}$ of formulas, in a flexible and compact way.
- **A compilation procedure** producing a differentiable computational graph from any LoH formula $\Phi$, allowing for the learning of a data-fitting logical formula among those in the hypothesis space represented by $\Phi$. We employ the Gödel trick (Daniele and van Krieken 2026), a newly proposed stochastic variant of Gödel logic. Thanks to this choice, our approach can directly learn discrete functions through backpropagation. Moreover, the computational graph can be stacked on top of a neural network, allowing the end-to-end learning of symbolic rules alongside perception-to-symbol mappings.
- **A unifying viewpoint** of NeSy paradigms. The general neural layers of rule-inducing NeSy models like (Wang et al. 2020; Payani and Fekri 2019) can be seen as the compilation of particular LoH formulas using product fuzzy logic. On the other hand, the full injection of prior knowledge can be obtained by simply avoiding the usage of the choice operator in a LoH formula. However, LoH is not limited to those two extremes and allows to construct models for many different intermediate situations (see Section 6).

Code and supplementary materials are available at: dbizzaro.github.io/LoH

## 2 Related Works

**Logics on top of Neural Predicates.** Approaches such as *SBR* (Diligenti, Gori, and Sacca 2017), *LTN* (Badreddine et al. 2022) and Semantic Loss (Xu et al. 2018) translate logical knowledge into differentiable penalties added to the loss. In contrast, abductive methods (Dai et al. 2019; Tsamoura, Hospedales, and Michael 2021; Huang et al. 2021) let a symbolic model find labels for the neural part consistent with the provided knowledge. This enables logical reasoning also at inference time, yet the symbolic program remains user-supplied rather than learned. Similarly, *DeepProbLog* (Manhaeve et al. 2018), *DeepStochLog* (Winters et al. 2022), and *NeurASP* (Yang, Ishay, and Lee 2020) enrich logical solvers with neural predicates whose outputs are treated as probabilities. Here what is learned sits on neural modules rather than on alternative rule bodies.

*Neuro-fuzzy networks* like (Jang 1993), (Zhou and Quek 1996) and (Nguyen, Zhou, and Quek 2015) are interpretable models where a neural network structure directly coincides with a fuzzy rule base. Typically they concentrate on parametric identification under fixed rules. Even when logic is induced (Shihabudheen and Pillai 2018), their rule-based architecture is suited for a DNF-like format, which contrast the expressivity and flexibility of LoH.

In probabilistic logic programming, probabilistic facts and annotated disjunctions could be employed to represent "choices". However, these define distributions over possible worlds, and extracting a single crisp program typically requires probabilities to collapse to $0/1$, as picking always the most likely local choice is only a heuristic. In LoH, instead, choices live in the space of *hypotheses*, and a compiled LoH formula act as one function in the hypothesis space, making binary deterministic yes/no predictions: the one obtained substituting each choice operator with the subformula with weight $> 0.5$. And learning can be seen as a differentiable combinatorial local search procedure under the *Gödel Trick* (Daniele and van Krieken 2026). This makes LoH especially suited for rule learning and deterministic decision modeling, whereas probabilistic logic programs are naturally tailored to probabilistic inference under uncertainty.

On the rule-inducing side, *SATNet* (Wang et al. 2019) embeds a smoothed MaxSAT layer inside a neural network, jointly optimizing clause weights and perception. Subsequent work exposed limitations with unsupervised grounding (Chang et al. 2020), partially alleviated in (Topan, Rolnick, and Si 2021). *DSL* (Daniele et al. 2022) directly learn symbolic rules from data alongside the perception-to-symbol mappings, but the symbolic part is only a lookup table.

**Neural Networks with Soft Gates.** Many recent NeSy learners devise neurons that perform a continuous relaxation of the `AND` and `OR` operations, with learnable weights acting as soft gates. These neurons are placed into layers alternating the two operations, while the `NOT` operation is obtained by doubling the inputs—juxtaposing each input value with its negation. Models like these are typically used to learn propositional rules on binarized tabular data (Qiao, Wang, and Lin 2021; Dierckx, Veroneze, and Nijssen 2023; Katzir, Elidan, and El-Yaniv 2020; Kusters et al. 2022; Perreault et al. 2025). Among these, *Multi-Layer Logical Perceptron (MLLP)* (Wang et al. 2020) is a state-of-the-art model, using product fuzzy logic operations. However, like all the others, it does not guarantee that the extracted rules—which it calls *Concept Rule Sets (CRS)*—have the same accuracy as the neural model. Instead, we propose to use Gödel fuzzy logic, which allows a lossless extraction.

Soft logical gates are also employed on binary/ternary neural networks (Deng et al. 2018), which are typically used for the efficiency of the quantized networks at inference, rather than the logical semantics. In particular, *Differentiable Logic Networks (DLN)* (Petersen et al. 2022; Petersen et al. 2024; Yousefi et al. 2025) keep a sparse fixed wiring with nodes having at most two parents, and learn which binary Boolean operation each node should execute. Instead, LoH does the opposite: the logical operations are fixed, and the learnable gates decide which branch is selected. This allows LoH to yield more readable formulas (as *DLNs* rely on deeply nested structures employing 16 different logical operators), and offers a more straightforward path for incorporating prior knowledge.

**Inductive Logic Programming (ILP).** Also modern NeSy methods in ILP, such as *∂ILP* (Evans and Grefenstette 2018), *NTP* (Campero et al. 2018) and *NeurRL* (Gao et al. 2025), use neurons performing a soft version of the logical operations, thus learning first-order rules via gradient descent. Of particular interest are *Logical Neural Networks* (Riegel et al. 2020), which compile formulas into neural networks with weighted Łukasiewicz operators, but rely on constrained optimization, which hampers scalability. Moreover, assigning "importances" to the subformulas, instead of choosing one among the candidates as in LoH, limits the possibility to extract hard rules without loss in accuracy.

In principle, ILP is a natural paradigm to span the full "zero to full knowledge" spectrum, since it can exploit background knowledge while inducing missing rules. However, as discussed in Section 6, current differentiable ILP systems do not implement this possibility: expressing intermediate regimes (e.g., selecting subsets of a given pool of rules or even just providing formulas not following the system templates) requires substantial ad-hoc machinery, and it is less practical and less transparent than LoH.

## 3 Background

Classical propositional logic builds formulas from propositional variables using negation ($\neg$), conjunction ($\wedge$) and disjunction ($\vee$).[1] An interpretation assigns to each variable the Boolean value true (1) or false (0), and extends recursively: the interpretation of $\neg\phi$ flips the value of $\phi$,

[1]For simplicity, we do not consider implication ($\rightarrow$) and iff ($\leftrightarrow$). However, there is nothing preventing their use, if associated with a consistent fuzzy semantics. For example, we may consider *material* implication, substituting $\phi \rightarrow \psi$ with its Boolean equivalent $\neg\phi \vee \psi$.

$\phi \wedge \psi$ returns the conjunction (AND) of the two values, and $\phi \vee \psi$ returns the disjunction (OR). Fuzzy logics relax the interpretations' truth values to the real unit interval $[0, 1]$, interpreting connectives with *t-norms* (for $\wedge$) and *t-conorms* (for $\vee$). This relaxation brings differentiable operations, allowing gradient-based optimization. Common fuzzy logics include Łukasiewicz, Product, and Gödel. Product logic has $t(x, y) := xy$ as t-norm (i.e., conjunction), and $s(x, y) := 1-(1-x)(1-y) = x+y-xy$ as t-conorm (i.e., disjunction). On the other hand, Gödel logic uses $\min$ and $\max$ for conjunction and disjunction, respectively. In both, the negation of $x$ corresponds to $1-x$.

Gödel logic stands out for its simplicity and its closer alignment to classical logic in terms of idempotency and distributivity. Importantly, Gödel logic has the following property:

**Theorem 1** (Prop. 1 in (Daniele and van Krieken 2026))**.** *For any Gödel interpretation $\mathcal{G}$, let $\mathcal{B}$ be the Boolean interpretation obtained rounding every fuzzy value in $\mathcal{G}$ with the thresholding function*[2]

$$\rho\colon [0,1] \setminus \{0.5\} \to \{0,1\}, \quad x \mapsto \begin{cases} 1 & \text{if } x > 0.5 \\ 0 & \text{if } x < 0.5 \end{cases} \tag{1}$$

*meaning that $\mathcal{B}(v_i) = \rho(\mathcal{G}(v_i))$ for every propositional variable $v_i$. Then, $\mathcal{B}$ is always consistent with $\mathcal{G}$, i.e., $\mathcal{B}(\phi) = \rho(\mathcal{G}(\phi))$ for every formula $\phi$.*

We can think at any logical formula as our model, with the truth values of the variables as inputs and the corresponding truth value of the formula as output. It is continuous if using fuzzy interpretations, and discrete if using classical Boolean ones. The theorem means that discretizing the outputs of the continuous model is the same as working with the discrete one, on the discretized inputs.

The main problem of using Gödel semantics is that its optimization can stall in shallow local minima. The Gödel Trick (Daniele and van Krieken 2026) counters this by adding noise to each parameter, turning the optimization into a stochastic local search while remaining gradient-based. It is equivalent to the the classical Gumbel-max trick (Gumbel 1954), and it works by storing as parameters the logits of the fuzzy weights. Then, for every step of training, random noise is sampled and added to the logits. This is done in the forward pass, before applying the sigmoid function producing the fuzzy weights.

## 4 A Language for Hypothesis Spaces

We introduce *Logic of Hypotheses* (LoH) first as a language for expressing hypothesis spaces (i.e., sets) of formulas in compact way. Syntactically, LoH extends propositional logic, adding a new *choice operator* $[\cdot]$ that can take as input any finite number of formulas:

$$\Phi ::= \top \mid \bot \mid v \mid \neg\Phi_1 \mid \Phi_1 \vee \Phi_2 \mid \Phi_1 \wedge \Phi_2 \mid [\Phi_1, \ldots, \Phi_n]$$

where $n$ can vary among the positive integers and $v \in \mathcal{V}$ represents the propositional variables. Semantically, a LoH formula $\Phi$ represents an entire set of classical propositional formulas $\mathcal{H}(\Phi)$, each obtained by selecting exactly one subformula per choice operator.

**Example 1.** *The LoH formula $\Phi := [a, b] \wedge [c, d] \wedge \neg e$ has hypothesis space*

$$\mathcal{H}(\Phi) := \{\, a \wedge c \wedge \neg e,\ a \wedge d \wedge \neg e,\ b \wedge c \wedge \neg e,\ b \wedge d \wedge \neg e \,\}$$

In general, the set $\mathcal{H}(\Phi)$ is obtained by applying inductively the following operations:

R1: $\mathcal{H}(v) = \{v\};\ \mathcal{H}(\top) = \{\top\};\ \mathcal{H}(\bot) = \{\bot\};$

R2: $\mathcal{H}(\neg\Phi_1) = \{\neg\phi \mid \phi \in \mathcal{H}(\Phi_1)\};$

R3: $\mathcal{H}(\Phi_1 \wedge \Phi_2) = \{\phi \wedge \psi \mid \phi \in \mathcal{H}(\Phi_1),\ \psi \in \mathcal{H}(\Phi_2)\};$

R4: $\mathcal{H}(\Phi_1 \vee \Phi_2) = \{\phi \vee \psi \mid \phi \in \mathcal{H}(\Phi_1),\ \psi \in \mathcal{H}(\Phi_2)\};$

R5: $\mathcal{H}([\Phi_1, \ldots, \Phi_n]) = \bigcup_{i=1}^{n} \mathcal{H}(\Phi_i).$

**Example 2.** *Let's unfold the step-by-step procedure for producing $\mathcal{H}([a, [b, c]] \wedge \neg[c, d])$:*

*(R1)* $\mathcal{H}(a)=\{a\}$; $\mathcal{H}(b)=\{b\}$; $\mathcal{H}(c)=\{c\}$; $\mathcal{H}(d)=\{d\}$;

*(R5)* $\mathcal{H}([b, c])=\{b, c\}$; $\mathcal{H}([c, d])=\{c, d\}$;

*(R5)* $\mathcal{H}([a, [b, c]])=\{a, b, c\}$;

*(R2)* $\mathcal{H}(\neg[c, d])=\{\neg c, \neg d\}$;

*(R3)* $\mathcal{H}([a, [b, c]] \wedge \neg[c, d])=\{a \wedge \neg c,\ a \wedge \neg d,\ b \wedge \neg c,\ b \wedge \neg d,\ c \wedge \neg c,\ c \wedge \neg d\}$

In neural networks, the output of a hidden neuron is usually fed to multiple neurons of the subsequent layer. Similarly, in LoH, we may want to use the same "choice" of a subformula in multiple places. This can be solved by defining a placeholder for a sub-formula, and use it in multiple parts of the main formula. The algorithm for producing the hypothesis space corresponding to an LoH formula with such placeholders is reported in Appendix A.

**Example 3.** *The LoH formula $[a, b] \wedge [a, b]$ has hypothesis space $\{a \wedge a, a \wedge b, b \wedge a, b \wedge b\} \equiv \{a, a \wedge b, b\}$. On the other hand, the LoH formula $\phi \wedge \phi$ with $\phi := [a, b]$ has hypothesis space $\{a \wedge a, b \wedge b\} \equiv \{a, b\}$. In $[a, b] \wedge [a, b]$ the two choices are independent, whereas $\phi \wedge \phi$ introduces just one choice and reuses the selected subformula twice.*

Notice that LoH is flexible enough to encode *any* finite set of propositional formulas $\{h_1, \ldots, h_n\}$. Indeed, $[h_1, \ldots, h_n]$ represents exactly that space, even if more compact representations—whose compilations will require less parameters—may be possible. Even for a fixed hypothesis space, LoH is flexible enough to provide formulas biasing the search process in different ways. For example, when compiled, both $[a, [b, c]]$ and $[a, a, b, c]$ are more biased towards choosing $a$ than $[a, b, c]$.

## 5 To Differentiable Computational Graphs

In the preceding section, we presented LoH as a language for expressing hypothesis spaces of logical formulas. We now show how the same LoH formulas can be turned into supervised machine learning models searching in those hypothesis spaces. The search will be done by gradient descent with backpropagation, so we need to compile LoH formulas into differentiable computational graphs.

[2]The exclusion of $x = 0.5$, where negation would break the homomorphism property, is mostly a technicality. In practical settings, this exact value has measure zero in continuous-valued interpretations and does not affect the general applicability of the result.

The first step is to introduce a weight $w_i \in [0,1]$ for every candidate subformula $\Phi_i$ inside a choice operator. These are learnable and act as gates. Each choice operator is then converted to a propositional formula linking such weights to the respective subformulas. This can be done in two dual, practically interchangeable ways, which we call *disjunctive/conjunctive compilations*:

$$\textit{Disjunctive Compilation} \quad [\Phi_1, \ldots, \Phi_n] \rightsquigarrow \bigvee_{i=1}^{n} w_i \wedge \Phi_i$$

$$\textit{Conjunctive Compilation} \quad [\Phi_1, \ldots, \Phi_n] \rightsquigarrow \bigwedge_{i=1}^{n} \neg w_i \vee \Phi_i$$

Whichever of the two we use—more on this later—, we are left with a propositional formula with only the operators $\neg$, $\wedge$ and $\vee$. In order to have differentiable operations, we interpret them under a fuzzy semantics. For example, with Gödel fuzzy logic, $\bigvee_{i=1}^{n} w_i \wedge \Phi_i$ becomes $\max_{i=1,\ldots,n}(\min(w_i, \Phi_i))$ and $\bigwedge_{i=1}^{n} \neg w_i \vee \Phi_i$ becomes $\min_{i=1,\ldots,n}(\max(1-w_i, \Phi_i))$. The final piece is to design the weights in such a way that they can take continuous values in the interval $[0,1]$, while allowing to extract the discrete selection of a candidate subformula for every choice operator.

**Design of the weights.** Let us first consider the case in which the weights are binary, i.e., each $w_i$ can only have value $0$ or $1$. Then, the formulas above can be simplified to equivalent ones, recalling that $0 \wedge \Phi \equiv 0$, $1 \wedge \Phi \equiv \Phi$, $0 \vee \Phi \equiv \Phi$ and $1 \vee \Phi \equiv 1$, for any formula $\Phi$. It follows that the disjunctive (resp. conjunctive) compilation is equivalent to the disjunction (resp. conjunction) of the subformulas with weight $1$. So if we impose that one weight $w_i$ is $1$ and the remaining are $0$, then both the conjunctive and the disjunctive compilation become equivalent to the single "chosen" subformula (the one with $w_i = 1$). This is exactly the condition we want after discretizing the weights: the discrete selection of a *single* candidate from each choice operator.

The simplest way to discretize the weights is to use the thresholding function $\rho$ defined in (1). Hence, for any tuple of weights $(w_1, \ldots, w_n) \in [0,1]^n$ associated to a choice operator $[\Phi_1, \ldots, \Phi_n]$, we want to impose that $w_i > 0.5$ for one and only one $i$. Instead of storing the weights $w_i$ directly, let us associate to each of them the actual learnable parameter $z_i$, which can take any real value. The differentiable operations for deriving the weights $w_i$ from these parameters are the following:

1. To escape local minima in the optimization procedure, at each *forward step of training*, add random noise to the parameters: $z'_i := z_i + n_i$ with $n_i \sim Gumbel(0, \beta)$ and $\beta$ being an hyperparameter.
2. Let $\bar{z}'$ be the mean of the two largest $z'_i$ values, and subtract it to each $z'_i$. By construction, all points $z'_i$ but the largest lie on the left of $\bar{z}'$.[3] Hence, one and only one among the shifted values $z''_i := z'_i - \bar{z}'$ is positive.
3. Apply the sigmoid function: $w_i := \sigma(z''_i/T) = \frac{\exp(z''_i/T)}{1+\exp(z''_i/T)}$, with the temperature $T$ being an hyperparameter.

The application of the sigmoid function ensures that the weights are in the interval $[0,1]$. Moreover, because of the second step, one and only one logit $z''_i$ is positive, so exactly one weight $w_i$ is greater than $0.5$. This procedure for producing weights $w_i$ from the $z_i$'s is an adaptation of the Gödel trick with categorical variables (Daniele and van Krieken 2026).

**Disjunctive vs Conjunctive Compilation.** Both the disjunctive and the conjunctive compilation have the same purpose of selecting *one* subformula among the candidates, and in general both work well. However, we suggest using the former when the choice operator is a term in a disjunction, and the latter when in a conjunction; the opposite if negated. For example, for $\neg[a,b] \wedge [b,c] \wedge ([c,d] \vee \neg[d,e])$, we suggest using disjunctive compilation for $[a,b]$ and $[c,d]$, and conjunctive compilation for $[b,c]$ and $[d,e]$. The theoretical and empirical motivations for this are discussed in the supplementary materials.

**Robustness to Binarization.** By design, it is always possible to binarize the weights of a model and extract the chosen subformula from each choice operator. The result is a propositional formula in the hypothesis space denoted by the LoH formula. If using Gödel fuzzy logic, then Theorem 1 guarantees that the outputs of the resulting propositional formula *coincide* with the rounding of the outputs of the continuous model on *every* possible input. Accuracy, confusion matrices, and any higher-level deductive tasks are thus preserved from the continuous to the discrete model. In this sense, Gödel logic offers lossless rule extraction (while Łukasiewicz and Product t-norms do not). Also notice that this is true at any point in training. So the entire training process—while being gradient-based—can be interpreted symbolically, through discrete changes. To our knowledge, no other rule-learning NeSy model achieves this.

## 6 LoH as a Unifying NeSy Framework

In this section, we demonstrate how LoH captures a wide range of existing NeSy frameworks, spanning fully-provided prior knowledge, partially known templates, and purely data-driven rule induction. To illustrate these settings, we instantiate them on a small synthetic task: a wildfire–risk assessment example problem.

**Experimental Setup: Wildfire Risk.** We generated a dataset of $2048$ samples, each consisting of a simulated aerial image and a set of 7 Boolean environmental features (Wind, LowHum, HighTemp, Rained, Lightnings, Isolated, PowerLines). The task is to predict a binary WFRisk label. To generate the images and the ground-truth WFRisk labels, two additional Boolean variables (Forest and DryVegetation)

[3] The probability of the two largest values coinciding is negligible, especially after adding the continuous Gumbel noise. Moreover, this happening would affect only the extraction of hard rules, not the optimization procedure (which would continue to change the parameters, eventually breaking the tie).

are used. Indeed, the ground-truth labels are generated as the conjunction of the following three rules:

$$\begin{aligned} \mathsf{Fuel} &:= \mathsf{Forest} \vee (\mathsf{DryVegetation} \wedge \mathsf{Wind}) && \text{(2a)} \\ \mathsf{Dry} &:= \mathsf{LowHum} \vee (\mathsf{HighTemp} \wedge \neg\mathsf{Rained}) && \text{(2b)} \\ \mathsf{Trigger} &:= \mathsf{Lightnings} \vee \neg\mathsf{Isolated} \vee \mathsf{PowerLines} && \text{(2c)} \\ \mathsf{WFRisk} &:= \mathsf{Fuel} \wedge \mathsf{Dry} \wedge \mathsf{Trigger} && \text{(2)} \end{aligned}$$

The model architectures consists of (i) a generic CNN $h_\theta$ that processes each image to predict Forest and DryVegetation as latent concepts, and (ii) a logical component that combines these visual concepts with the environmental features to produce the final prediction. Crucially, the CNN is not pretrained: it must learn to identify forests and dry vegetation solely via the gradients backpropagated through the logic. We now analyze how LoH handles this task under different knowledge assumptions.

**Full Knowledge (No Choice Operators).** If the choice operator $[\cdot]$ is never used, the logical part has no learnable parameter, but the gradient can backpropagate from it to a neural part below. This corresponds to the NeSy approaches where knowledge is entirely specified a priori.

For the wildfire experiment, we set the LoH formula exactly to the ground-truth rule (2). Since the logic is fixed and correct, the learning focuses entirely on the perception module. The model successfully solves this symbol grounding problem, learning to map images to Forest and DryVegetation with high accuracy simply by minimizing the classification error of the WFRisk label (see Figure 1).

**Selecting Reliable Rules.** Suppose we have a knowledge set of $n$ candidate rules $r_1, \ldots, r_n$, but we are unsure on whether they are correct. Then, we can use the following LoH formula to select a reliable subset:

$$\bigwedge_{i=1}^{n} [r_i, \top] \tag{3}$$

Indeed, the hypothesis space spans all possible subsets: by selecting $r_i$ over $\top$, the model effectively picks such rule. When the rules are clauses (i.e., disjunctions of possibly negated propositions), this setup parallels *KENN* (Daniele and Serafini 2019), which also have a learnable weight for each rule (even if it is never made discrete). In our framework, each $r_i$ in (3) can be any formula. For example, we may have entire knowledge bases as $r_i$'s, and use (3) to decide which to trust.

On the wildfire task, we build a pool of rules by taking the three ground-truth rules Fuel, Dry and Trigger and augmenting them with 12 plausible but incorrect variants (see Appendix B). Formula (3) is then applied to this pool. The resulting learned subset is a data-driven refinement of the original possibilities.

**Selecting One Rule per Set.** Given $m$ sets $\{r_{i,1}, \ldots, r_{i,n_i}\}$ of candidate rules, we may want to enforce the choice of *exactly one* rule per set. This may happen for example because the rules in each set are mutually exclusive. For this purpose, we can use the following LoH formula:

$$\bigwedge_{i=1}^{m} [r_{i,1},\, r_{i,2}, \ldots, r_{i,n_i}] \tag{4}$$

For wildfire risk, suppose domain experts agree on the structure of the final rule as a conjunction of three components—fuel, dryness, trigger—but propose several alternatives for each component. We group these into three sets and apply (4) with $m = 3$: one choice among fuel candidates, one among dryness candidates, and one among trigger candidates (see Appendix B). This regime differs from the previous one in that the model must commit to a single alternative within each group, instead of freely activating or deactivating rules.

**Partial Knowledge Base.** If we have a knowledge base $K$ which is reliable but not complete, we can couple it with a rule-learning LoH formula $\Phi$, and use $K \wedge \Phi$. Similarly, we may have a knowledge base whose rules have some missing parts, and fill them with some rule-learning formulas.

As an example, in wildfire risk assessment, we consider the Fuel rule as given, while the rest is unknown and must be selected from the non-fuel candidate rules used in the previous settings (see Appendix B). For these candidates, we apply the subset-selection scheme of (3), so that LoH learns which dryness- and trigger-related rules best complement the known Fuel rule.

**Results and Comparison to ILP.** Figure 1 compares the four knowledge regimes introduced so far: full knowledge, selecting reliable rules, selecting one rule per set, and partial knowledge base with Fuel given. As expected, richer priors yield better performance.

We also wanted to compare LoH against inductive logic programming (ILP). Among the differentiable ILP systems, we found no implementation that supports all regimes directly, by lacking a way to constrain the hypothesis spaces to operations such as "select a subset of given rules". We therefore added the required machinery starting from a $\partial$ILP (Evans and Grefenstette 2018) public implementation,[4] and used such modified system as a baseline.[5] After the changes, $\partial$ILP assigns a learnable weight to each candidate rule through its clause-selection parameters. This effectively reproduce a choice mechanism under product fuzzy logic, with similar degrees of freedom as LoH, but in a more cumbersome and less direct way.

Notably, in regimes where there is actually a logical part to be learned, LoH outperforms $\partial$ILP, with the performance

[4] https://github.com/ai-systems/DILP-Core

[5] Concretely, we (i) added a *mode-declaration*-style filter to restrict clause generation to prescribed sets (e.g., in "selecting one rule per set", Fuel/DryVegetation/Trigger can be defined only by one of their respective candidate rules); (ii) implemented the conjunction of multiple formulas via a conjunction structure with auxiliary predicates; (iii) grounded each candidate rule as a *differentiable extensional predicate* using product fuzzy logic (since defining the rules intensionally would break the clause templates); and (iv) added mini-batch training, exploiting sample independence.

gap widening in the settings involving the largest hypothesis spaces of logical rules (Selecting Reliable Rules, and Partial Knowledge Base).

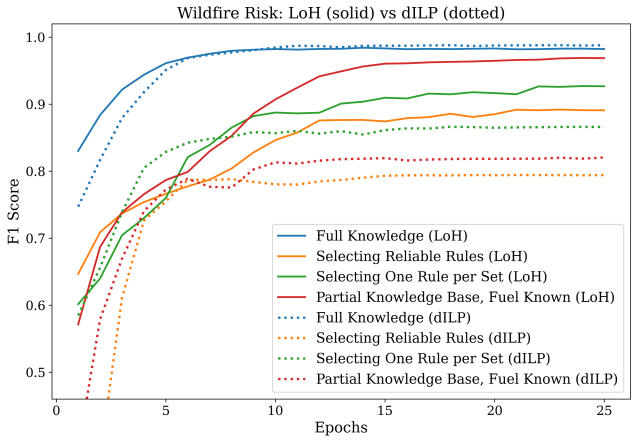



Figure 1: Average training curves, over 20 runs, of the LoH and ∂ILP models based on different levels of knowledge.

**Zero Knowledge (Pure Rule Learning).** For rule induction, we can combine neurons learning disjunctions and neurons learning conjunctions. A simple and most efficient way is to arrange them in layers and exploit parallel computation on tensors, like in standard Artificial Neural Networks. The following are LoH formulas for neurons learning the disjunction of a subset of neurons of the previous layer, with and without negations:

$$n_j^{(l+1)} := \bigvee_{i=1}^{m_l} [n_i^{(l)}, \neg n_i^{(l)}, \bot], \tag{5}$$

$$n_j^{(l+1)} := \bigvee_{i=1}^{m_l} [n_i^{(l)}, \bot] \tag{6}$$

Analogously, conjunctive neurons simply replace disjunction with conjunction and $\bot$ with $\top$. The layers adopted in NLNs (Payani and Fekri 2019) and MLLPs (Wang et al. 2020) use neurons analogous to those, with product fuzzy logic instead of Gödel's. In this correspondence, they would use—as we suggested—conjunctive compilation for the choice operators in conjunctive neurons and vice versa for disjunctive neurons. However, LoH is flexible and many alternative designs of neurons are possible. For example,

$$n_j^{(l+1)} := \bigvee_{i=1}^{k} [n_1^{(l)}, \ldots, n_{m_l}^{(l)}, \neg n_1^{(l)}, \ldots, \neg n_{m_l}^{(l)}], \tag{7}$$

$$n_j^{(l+1)} := \bigvee_{i=1}^{k} [n_1^{(l)}, \ldots, n_{m_l}^{(l)}] \tag{8}$$

are neurons learning clauses of fixed width $k$ (possibly with repetitions)—$k$ being a hyperparameter. A detailed empirical study of this zero-knowledge regime is deferred to Section 7.

**Respecting Syntactic Requirements.** If the learned rules of a NeSy model are to be used also in a symbolic program, this may require them to adhere to a specific format or template. If it is possible to express the template in LoH, and LoH is quite flexible, then the adherence is guaranteed. As an example, here is a possible template for clauses of width 3:

$$[v_1, \ldots, v_n, \neg v_1, \ldots, \neg v_n] \vee [v_1, \ldots, v_n, \neg v_1, \ldots, \neg v_n] \vee [v_1, \ldots, v_n, \neg v_1, \ldots, \neg v_n] \tag{9}$$

And here is a template for definite clauses (i.e., clauses with exactly one positive literal):

$$\bigvee_{i=1}^{n} [\neg v_i, \bot] \vee [v_1, \ldots, v_n] \tag{10}$$

## 7 Experiments

In this section, we evaluate the performance of our models, in their general form when no domain knowledge is provided: i.e., models made alternating conjunctive and disjunctive layers, with neurons of the form in (6) or in (8). We employ the suggested compilations with the Gödel trick, and test the performance on several benchmark classification datasets.

### 7.1 Classification on Tabular Datasets

**Datasets.** We use 12 classification benchmarks from the UCI Machine Learning Repository, available with CC-BY 4.0 license. They were previously employed in (Wang et al. 2020) for evaluating MLLP/CRS, a state-of-the-art rule-learning model using product fuzzy logic. As a preprocessing step, we adopt the same data discretization and binarization procedure as in (Wang et al. 2020): the recursive minimal entropy partitioning algorithm (Dougherty, Kohavi, and Sahami 1997), followed by one-hot-encoding.

**Models.** In our model, we alternate conjunctive and disjunctive layers without negations. The choice between using neurons of type (6) or neurons of type (8) is performed by the hyperparameters' tuning, together with the rest of the architecture (number and size of layers, and whether to start with a conjunctive or a disjunctive layer). We compare our model against Differentiable Logic Networks (DLN) (Petersen et al. 2022), the aforementioned MLLP (Wang et al. 2020), and the rules extracted from it, which are called CRS. Note that for DLN we report the performance of the continuous model, which is generally higher than the discretized one. Instead, Gödel semantics ensures our model behaves identically before and after binarization, and CRS corresponds to the post-hoc binarization of MLLP. We also consider commonly used machine learning baselines: Decision Trees (DT), Random Forests (RF), XGBoost and standard fully-connected Neural Networks (NN).

Five of the twelve datasets have more than two classes, and we want to treat multi-class predictions as mutually exclusive output propositions. Therefore, in our model, we apply a reparameterization to the final layer, to guarantee that exactly one output per example exceeds the threshold value

$0.5$—the output of the predicted class. This is analogous to the procedure used before for designing the weights of a choice operator, but does not require the addition of noise. Concretely, let $z_i$ be the logits of the outputs $o_i$ produced by the outmost layer—each one corresponding to a different class. Each logit gets shifted by subtracting the mean $\bar{z}$ of the two largest $z_i$'s. The resulting $z_i - \bar{z}$ values—of which, by construction, one and only one is positive—are then outputted through a sigmoid activation.

**Methodology.** Each dataset is divided into $20\%$ for testing and $80\%$ for training and validation. In particular, validation takes $12.5\%$ of the second split, so $10\%$ of the whole dataset. Since most datasets are unbalanced, we use the F1 score (macro) as classification metric. As loss functions, we use Binary Cross-Entropy for NN, DLN and our model, and Mean Squared Error for MLLP. All of them are optimized using Adam (Kingma 2014). For each dataset, the hyperparameters of each model are tuned using $80$ trials of the TPE algorithm (Bergstra et al. 2011) from the Optuna library. Architectural choices—such as the number and width of layers—are tuned alongside all other hyperparameters.

**Discussion.** For the selected hyperparameters, Table 1 provides the means and standard deviations of the test-set F1 scores, out of $10$ different training runs on the training plus validation sets. Although vanilla neural networks attain the best summary score ($.89$) and rank ($2.9$), they—together with RF and XGBoost—do not allow to extract symbolic knowledge. MLLP—which come second—does support rule extraction, but with the loss in performance from it to CRS. In fact, only DT, CRS, the discretization of DLN, and our model achieve all the benefits of symbolic discrete rules, such as better interpretability, explainability, and efficiency of inference. Moreover, unlike DT, RF and XGBoost, our model is fully differentiable, so it could be placed downstream of perception modules (CNNs, transformers, etc.) and be trained end-to-end.

Apart from the two datasets with the largest numbers of classes, namely *chess* (with 18) and *letRecog* (with 26), our model have consistently better scores than CRS, and is almost always on par with, or close to, the neural network baseline. This suggests that our handling of many-way classification may not be optimal. Notably, DLN struggles even more on those two benchmarks, and in general does not surpass our model even in its continuous (pre-discretization) form. Hence, when only a few classes are present, our model reliably ranks among the top performers while simultaneously providing symbolic rules and end-to-end differentiability.

## 7.2 Visual Tic-Tac-Toe

One of the previous classification benchmarks (Aha 1991) is based on the game of tic-tac-toe. The dataset provides all possible ending board configurations (for games in which $X$ begins), and the target is to predict whether $X$ has won the game or not. So the target function can be expressed with a simple formula in Disjunctive Normal Form (DNF), having $8$ clauses.[6] Let us introduce a more challenging variant of the dataset, which we refer to as *Visual Tic-Tac-Toe*. In this version, instead of symbolic board encodings, each cell is represented by a MNIST image. Specifically, we assign images of the digit $0$ to represent $X$, images of the digit $1$ to represent $O$, and blank cells are represented by images of the digit $2$. As a result, instead of a structured propositional encoding, the inputs consist of $3 \times 3$ grids of grayscale images. This modification significantly increases the difficulty of the task, as models must now also learn to recognize digit representations from the images, but with as little supervision as before (just the "$X$ winning" label). Traditional symbolic models would struggle with such high-dimensional and noisy input, making this an interesting benchmark for NeSy learning.

**Models.** To handle the image-based input, we extend all models with a standard CNN. This network consists of two convolutional layers, each using a kernel size of $3$ with padding $1$, followed by ReLU activations and max-pooling. The first convolutional layer has $32$ output channels, while the second has $64$. After feature extraction, the CNN flattens the output and passes it through a fully connected layer of size $128$, a dropout layer with probability $0.5$, and another fully connected layer of size $3$. The three outputs for each of the nine images composing a grid are then concatenated.

For DLN, MLLP/CRS and our models, the outputs of the CNN for the nine images are to be considered as input "propositions", so their values are clipped between $0$ and $1$. The learning process is end-to-end, including the CNN component. Crucially, the CNN is not pretrained to recognize the digits, nor are its outputs predefined to represent anything specific. The hyperparameters (and their tuning) are as before, but we allow the CNN part to have a separate learning rate (in the range $10^{-5}$–$10^{-3}$). Moreover, for both MLLP/CRS and our model, we fix the number of layers to 2 and distinguish the two cases of whether the last one is disjunctive (DNF) or conjunctive (CNF). This is not possible for DLN.

We also implemented a neural baseline that follows a classical deep learning approach (NN). This model uses a CNN module as the one described earlier, but its outputs are not clipped, and their number is a hyperparameter in the range 3-32 (instead of being fixed to 3). This is because in this case the outputs of this part of the network may be better seen as embeddings rather than symbols. The concatenation of such vectors (for the nine images in a grid) is then fed to a multi-layer perceptron.

**Methodology.** The original symbolic tic-tac-toe dataset is split with the same proportions as before ($70\%$-$10\%$-$20\%$). The training and validation parts are given images from MNIST training set, while the test part is given images from MNIST test set. No image is used more than once. Moreover,

[6] Let the input data be encoded with propositions $X_1$-$X_9$, $O_1$-$O_9$ and $B_1$-$B_9$, meaning that $X_i$ is true if there is an $X$ at position $(\lfloor i/3 \rfloor, i\%3)$, and similarly for $O$s and $B$lank cells. Then, the target function is $(X_1 \wedge X_2 \wedge X_3) \vee (X_4 \wedge X_5 \wedge X_6) \vee (X_7 \wedge X_8 \wedge X_9) \vee (X_1 \wedge X_4 \wedge X_7) \vee (X_2 \wedge X_5 \wedge X_8) \vee (X_3 \wedge X_6 \wedge X_9) \vee (X_1 \wedge X_5 \wedge X_9) \vee (X_3 \wedge X_5 \wedge X_7)$.

| Dataset | DT | RF | XGBoost | NN | MLLP | CRS | DLN | LoH |
|---|---|---|---|---|---|---|---|---|
| adult | .80 ±.00 | .81 ±.00 | **.82** ±.00 | .81 ±.01 | .80 ±.01 | .75 ±.06 | .80 ±.02 | **.81** ±.01 |
| bank-m. | .74 ±.00 | .73 ±.00 | .76 ±.00 | **.78** ±.01 | .69 ±.08 | .75 ±.01 | **.76** ±.01 | **.76** ±.01 |
| banknote | .95 ±.00 | .95 ±.01 | **.96** ±.00 | **.96** ±.00 | **.96** ±.00 | **.96** ±.00 | .95 ±.01 | **.96** ±.00 |
| blogger | .64 ±.00 | .53 ±.00 | .69 ±.00 | .82 ±.05 | **.84** ±.00 | .78 ±.02 | .72 ±.12 | **.83** ±.08 |
| chess | .81 ±.00 | .58 ±.01 | **.85** ±.00 | .82 ±.01 | .83 ±.02 | **.74** ±.02 | .41 ±.02 | .69 ±.01 |
| connect-4 | .59 ±.00 | .55 ±.00 | **.71** ±.00 | **.71** ±.04 | .58 ±.01 | .58 ±.01 | **.62** ±.01 | .58 ±.01 |
| letRecog | .80 ±.00 | .76 ±.00 | .92 ±.00 | **.93** ±.00 | .85 ±.01 | **.80** ±.01 | .64 ±.02 | .77 ±.01 |
| magic04 | .81 ±.00 | .83 ±.00 | **.84** ±.00 | **.84** ±.00 | **.84** ±.00 | .80 ±.00 | **.83** ±.00 | **.83** ±.00 |
| mushroom | **1.** ±.00 | **1.** ±.00 | **1.** ±.00 | **1.** ±.00 | **1.** ±.00 | **1.** ±.01 | **1.** ±.00 | **1.** ±.00 |
| nursery | .79 ±.00 | .79 ±.00 | .80 ±.00 | **1.** ±.00 | **1.** ±.00 | **1.** ±.00 | **1.** ±.00 | **1.** ±.00 |
| tic-tac-toe | .89 ±.00 | .98 ±.01 | .97 ±.00 | .99 ±.01 | **1.** ±.00 | **1.** ±.00 | .99 ±.01 | .99 ±.01 |
| wine | **1.** ±.00 | **1.** ±.01 | .96 ±.00 | .97 ±.05 | **1.** ±.00 | .96 ±.01 | .97 ±.02 | **.98** ±.01 |
| mean (↑) | .82 | .79 | .86 | **.89** | .87 | .84 | .81 | **.85** |
| avg. rank (↓) | 5.7 | 6.0 | 3.7 | **2.9** | 3.5 | 5.0 | 5.2 | **4.1** |

Table 1: Mean F1 scores on the test sets of tabular benchmarks, out of 10 runs. Rightmost models are highlighted separately because they combine differentiable learning with symbolic rule extraction (DLN is reported in its soft form but can be discretized; CRS is the discretization of MLLP).

for each board configuration in the training and validation sets, two different image grids are created, effectively doubling the number of samples.

Beyond performance, a key advantage of NeSy approaches lies in their better interpretability. To interpret the learned decision rules, we first want to assign meaningful labels to the propositions (that corresponds to the CNN outputs units, for each position in the $3 \times 3$ grid). After the training, we therefore inspect the three output units of the CNN, and determine whether a unit consistently responds to MNIST digits representing $X$, $O$, or blank cells. We assign $X$, $O$, or $B$ symbols based on such firing behavior, and use such names to extract and evaluate the learned logical formulas.

**Discussion.** Table 2 provides the mean and standard deviations of the test-set F1 scores, based on training each model 30 times on the combined training and validation sets. What changes between the runs is the network initialization, the mini-batches, the added noise in our model and the random binarization occurring in MLLP as a form of regularization. When possible, we report also the performance of the extracted formulas (Symbolic eval).

On 2 out of its 30 training runs, the neural baseline remained stuck at always predicting the most frequent class, with macro F1 score of .39. Instead, on the other runs, its performance was comparable to that of our CNF model. Similarly, MLLP/CRS in the DNF setting remained stuck on 4 runs, with the remaining 26 performing slightly worse than our CNF model. Finally, MLLP in the CNF version, and DLN, have similar performance to our CNF model on all runs. However, the symbolic evaluation reveals that DLN fails when discretized, and MLLP/CRS learned less accurate symbolic rules than ours. Moreover, the DNF version of our model is consistently better than all other models, and also found the 100%-correct formula reported above on 4 occasions. These results highlight the ability of our models to recover symbolic decision rules with high fidelity, even when the symbols must be learned from continuous high-dimensional perceptions.

## 8 Conclusion and Future Work

We introduced a single, compact language for expressing hypothesis spaces of logical formulas and compiling them into differentiable models whose discrete rule extraction is provably loss-free under Gödel semantics. LoH unifies knowledge injection and rule induction within one propositional paradigm, and yields strong results on both tabular and perceptual benchmarks, while retaining the possibility to extract learned logical formulas following arbitrary templates.

Despite these encouraging results, the present work leaves two important avenues open. First, the empirical validation should be broadened, targeting larger datasets, use of partial knowledge and additional NeSy tasks with complex perceptions. Second, LoH is currently propositional, while many NeSy applications are relational. In practice, many first-order systems already ground rules over a finite domain and then operate propositionally; the genuinely first-order aspect is the quantifier-driven aggregation over groundings. A first-order LoH could internalize this by adding $\forall$ and $\exists$ to the language, interpreting them as conjunctions/disjunctions over substitutions, and allowing choice operators to range over quantified and quantifier-free formulas alike. The main challenge would remain scalability due to the combinatorial blow-up of groundings, as in many FOL-based NeSy approaches.

## A LoH with Placeholders: a Formalization

We allow the user to *name* any LoH subformula by writing a declaration of the form $p \coloneqq \phi$, where $\phi$ is a LoH formula and $p$ is a fresh identifier (i.e., a name different from any propositional variable and any other identifier). A placeholder can itself mention other placeholders that have been declared earlier (the directed graph of such references must be acyclic,

| | | | MLLP | | CRS | | LoH | |
|---|---|---|---|---|---|---|---|---|
| | NN | DLN | DNF | CNF | DNF | CNF | DNF | CNF |
| NeSy eval | .91 ±.14 | .96 ±.01 | .80 ±.22 | .94 ±.02 | .76 ±.28 | .92 ±.00 | **.97 ±.01** | .95 ±.01 |
| Symbolic eval | - | .37 ±.21 | - | - | .80 ±.30 | .97 ±.02 | **.99 ±.00** | **.99 ±.00** |

Table 2: Comparison of the models on the Visual Tic-Tac-Toe task.

to avoid circular definitions).[7] Given a well-defined list of declarations $\mathcal{D} = \{p_1{:=}\phi_1, \ldots, p_m{:=}\phi_m\}$, Algorithm 1 produces the hypothesis space $\mathcal{H}(\Phi)$ for every LoH formula $\Phi$ possibly containing the placeholders $p_1, \ldots, p_m$.

**Algorithm 1** Construction of the hypothesis space $\mathcal{H}(\Phi)$

**Input:** LoH formula $\Phi$; declaration list $\mathcal{D} = \{p_i := \phi_i\}_{i=1}^m$
**Output:** $\mathcal{H}(\Phi)$, the hypothesis space of $\Phi$

1: **function** HYPOTHESES($\Phi, \mathcal{D}$)
2: **Tagging.** Traverse every choice node $[\Psi_1, \ldots, \Psi_n]$ in both $\Phi$ and the bodies $\phi_i$; assign a fresh identifier $c$ and arity $n_c$ to each.
3: **Indices Space.** $\mathcal{S} \leftarrow \prod_c \{1, \ldots, n_c\}$ (Cartesian pr.)
4: **Evaluator.** Define the function $\mathrm{E}(\cdot, \mathcal{I})$ recursively:

$$\mathrm{E}(v, \mathcal{I}) := v \qquad \text{(R1')}$$
$$\mathrm{E}(\neg\Psi_1, \mathcal{I}) := \neg\mathrm{E}(\Psi_1, \mathcal{I}) \qquad \text{(R2')}$$
$$\mathrm{E}(\Psi_1 \wedge \Psi_2, \mathcal{I}) := \mathrm{E}(\Psi_1, \mathcal{I}) \wedge \mathrm{E}(\Psi_2, \mathcal{I}) \qquad \text{(R3')}$$
$$\mathrm{E}(\Psi_1 \vee \Psi_2, \mathcal{I}) := \mathrm{E}(\Psi_1, \mathcal{I}) \vee \mathrm{E}(\Psi_2, \mathcal{I}) \qquad \text{(R4')}$$
$$\mathrm{E}([\Psi_1, \ldots, \Psi_n]_c, \mathcal{I}) := \mathrm{E}(\Psi_{\mathcal{I}[c]}, \mathcal{I}) \qquad \text{(R5')}$$
$$\mathrm{E}(p_i, \mathcal{I}) := \mathrm{E}(\phi_i, \mathcal{I}) \qquad \text{(Placeholders)}$$

5: **Enumeration.** $\mathcal{H}(\Phi) \leftarrow \{\mathrm{E}(\Phi, \mathcal{I}) \mid \mathcal{I} \in \mathcal{S}\}$
6: **return** $\mathcal{H}(\Phi)$
7: **end function**

Given $\Phi$ and $\mathcal{D}$, the Cartesian product $\mathcal{S}$ (in line 3) is exactly the set of all possible assignments of concrete choices to every choice operator. Indeed, for every $\mathcal{I} \in \mathcal{S}$, the algorithm considers an index $\mathcal{I}[c] \in \{1, \ldots, n_c\}$ for every choice node $c$ (of arity $n_c$), and the evaluator $\mathrm{E}(\cdot, \mathcal{I})$ is a function that (i) respects placeholder sharing and (ii) replaces every choice node $[\Psi_1, \ldots, \Psi_{n_c}]_c$ with the branch $\Psi_{\mathcal{I}[c]}$. Thus, $\mathrm{E}(\Phi, \mathcal{I})$ reflects the discrete model obtained compiling $\Phi$ (as in Section 5), when $\mathcal{I}[c]$ is the index of the unique weight greater than $0.5$, for each compiled choice operator $c$. It follows that $\mathcal{H}(\Phi)$ coincides exactly with the hypothesis space of the compiled and discretized model, as we wanted.

When compiled, placeholders implement *weight sharing*: all occurrences of the same placeholder reuse the same choice parameters, so a single shared selection is made wherever the placeholder is referenced.

[7] Notice that one can still build LoH-based recurrent networks exactly as one builds ordinary RNNs: by passing the hidden state from one time step to the next. In this case, one may write an LoH formula formally relating some placeholders to the ones at the previous time step, and acyclicity is in the unrolled architecture.

# B Wildfire Risk: Candidate Rules

Section 6 compares several ways of exploiting symbolic knowledge. The task is defined over nine propositional variables. Two of them are *visual* concepts, to be formed by a CNN from the images: Forest and DryVegetation (see Figure 2). The remaining seven are *non-visual* features provided in tabular form: LowHum, HighTemp, Rained, Wind, Lightnings, Isolated and PowerLines. The ground-truth label WFRisk is obtained by combining three interpretable intermediate rules: see Equations (2).

Across the knowledge regimes, we start from three sets of candidate rules (for Fuel, Dryness, and Trigger), each comprising one ground-truth rule and a set of distractors. We index candidates as $\phi_{i,j}$, where $i \in \{\mathsf{Fuel}, \mathsf{Dry}, \mathsf{Trigger}\}$ and $j \in \{1, \ldots, n_i\}$. The ground truth rules are those with $j = 1$.

- **Fuel candidates** ($n_{\mathsf{Fuel}} = 4$):

$$\phi_{\mathsf{Fuel},1} := \mathsf{Forest} \vee (\mathsf{DryVegetation} \wedge \mathsf{Wind})$$
$$\phi_{\mathsf{Fuel},2} := \mathsf{Forest}$$
$$\phi_{\mathsf{Fuel},3} := \mathsf{Forest} \wedge \mathsf{DryVegetation}$$
$$\phi_{\mathsf{Fuel},4} := \mathsf{DryVegetation} \wedge (\mathsf{Wind} \vee \mathsf{LowHum})$$

- **Dryness candidates** ($n_{\mathsf{Dry}} = 6$):

$$\phi_{\mathsf{Dry},1} := \mathsf{LowHum} \vee (\mathsf{HighTemp} \wedge \neg\mathsf{Rained})$$
$$\phi_{\mathsf{Dry},2} := \mathsf{LowHum} \wedge \mathsf{HighTemp} \wedge \neg\mathsf{Rained}$$
$$\phi_{\mathsf{Dry},3} := \mathsf{HighTemp} \vee (\mathsf{LowHum} \wedge \neg\mathsf{Rained})$$
$$\phi_{\mathsf{Dry},4} := \neg\mathsf{Rained} \vee (\mathsf{LowHum} \wedge \mathsf{Wind})$$
$$\phi_{\mathsf{Dry},5} := \neg\mathsf{Rained} \vee (\mathsf{LowHum} \wedge \mathsf{HighTemp})$$
$$\phi_{\mathsf{Dry},6} := \mathsf{DryVegetation} \wedge \neg\mathsf{Rained}$$

- **Trigger candidates** ($n_{\mathsf{Trigger}} = 5$):

$$\phi_{\mathsf{Trigger},1} := \mathsf{Lightnings} \vee \neg\mathsf{Isolated} \vee \mathsf{PowerLines}$$
$$\phi_{\mathsf{Trigger},2} := \mathsf{Lightnings} \wedge \mathsf{Isolated}$$
$$\phi_{\mathsf{Trigger},3} := \mathsf{PowerLines} \wedge \mathsf{Wind}$$
$$\phi_{\mathsf{Trigger},4} := \mathsf{Rained} \wedge \mathsf{Lightnings}$$
$$\phi_{\mathsf{Trigger},5} := \neg\mathsf{Isolated}$$

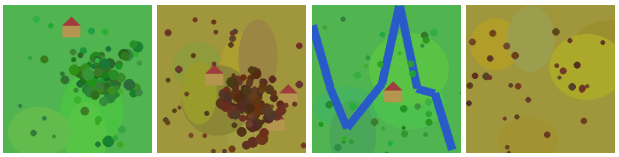
Let $\mathcal{I} := \{\mathsf{Fuel}, \mathsf{Dry}, \mathsf{Trigger}\}$. In the following we report the LoH formula used for each knowledge regime.

- **Full Knowledge:**

$$\phi_{\mathsf{Fuel},1} \;\wedge\; \phi_{\mathsf{Dry},1} \;\wedge\; \phi_{\mathsf{Trigger},1}$$

- **Selecting Reliable Rules:**

$$\bigwedge_{i \in \mathcal{I}} \bigwedge_{j=1}^{n_i} [\, \phi_{i,j},\ \top \,]$$



(a) $F, \neg DV$ (b) $F, DV$ (c) $\neg F, \neg DV$ (d) $\neg F, DV$

Figure 2: Examples of synthetic wildfire images for the four possible combinations of Forest ($F$) and DryVegetation ($DV$).

- **Selecting One Rule per Set:**

$$\bigwedge_{i \in \mathcal{I}} [\, \phi_{i,1},\, \phi_{i,2},\, \ldots,\, \phi_{i,n_i}]$$

- **Partial Knowledge Base, Fuel Known:**

$$\phi_{\mathsf{Fuel},1} \;\wedge \bigwedge_{i \in \mathcal{I} \setminus \{\mathsf{Fuel}\}} \bigwedge_{j=1}^{n_i} [\, \phi_{i,j}, \top\,]$$

## C Formulas Length

To evaluate the interpretability in terms of syntactic compactness, we compared the length of the propositional formulas extracted from CRS and LoH. We restricted this analysis to the binary classification datasets of Section 7.1. To ensure a fair comparison, we constrained the architectures to yield a formula in disjunctive normal form (DNF) by design: a single hidden conjunctive layer encoding candidate clauses, followed by a disjunctive output neuron. Except for this architectural constraint, all other hyperparameters and protocol details were tuned exactly as in the main experiment. Table 3 reports the number of clauses and the mean clause width, averaged over 10 independent runs. Before computing such values, we applied a simple pruning procedure following (Wang et al. 2020): we removed clauses that subsume others and are therefore redundant, and we discarded clauses that were never satisfied by any instance in the training set.

| Dataset | CRS | | LoH | |
|---|---|---|---|---|
| | #Clauses | Cl. Size | #Clauses | Cl. Size |
| adult | 10.8 | 2.4 | 17.3 | 3.4 |
| bank-m. | 118.8 | 6.1 | 14.3 | 4.8 |
| banknote | 8.3 | 1.9 | 10.5 | 2.5 |
| blogger | 6.6 | 2.8 | 9.1 | 3.1 |
| magic04 | 73.7 | 2.7 | 30.6 | 2.8 |
| mushroom | 7.7 | 3.9 | 10.0 | 7.6 |
| tic-tac-toe | 8.0 | 3.0 | 12.2 | 3.5 |

Table 3: Average number of clauses and average clause size of the extracted DNF formulas, over 10 runs.

Overall, we find that CRS tends to produce smaller formulas than LoH in many simple cases. However, for *bank-marketing* (Moro, Rita, and Cortez 2014) and *magic04* (Bock 2004), the extracted CRS formulas are extremely long outliers. In contrast, LoH does not exhibit such explosions in formula length.

## AI Declaration

We used LLMs as general-purpose assist tools for polishing prose, for minor code completions, and for generating the code used to produce the synthetic wildfire images. All such outputs were reviewed and verified by the authors, who take full responsibility.

## References

Aha, D. 1991. Tic-Tac-Toe Endgame. UCI Machine Learning Repository.

Badreddine, S.; Garcez, A. d.; Serafini, L.; and Spranger, M. 2022. Logic tensor networks. *Artificial Intelligence* 303:103649.

Barbiero, P.; Ciravegna, G.; Giannini, F.; Espinosa Zarlenga, M.; Magister, L. C.; Tonda, A.; Lio, P.; Precioso, F.; Jamnik, M.; and Marra, G. 2023. Interpretable neural-symbolic concept reasoning. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, 1801–1825.

Bergstra, J.; Bardenet, R.; Bengio, Y.; and Kégl, B. 2011. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc.

Besold, T. R.; Bader, S.; Bowman, H.; Domingos, P.; Hitzler, P.; Kühnberger, K.-U.; Lamb, L. C.; Lima, P. M. V.; de Penning, L.; Pinkas, G.; et al. 2021. Neural-symbolic learning and reasoning: A survey and interpretation. In *Neuro-Symbolic Artificial Intelligence: The State of the Art*. IOS press. 1–51.

Bock, R. 2004. MAGIC Gamma Telescope. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C52C8B.

Campero, A.; Pareja, A.; Klinger, T.; Tenenbaum, J.; and Riedel, S. 2018. Logical rule induction and theory learning using neural theorem proving. *arXiv preprint arXiv:1809.02193*.

Chang, O.; Flokas, L.; Lipson, H.; and Spranger, M. 2020. Assessing satnet's ability to solve the symbol grounding problem. *Advances in Neural Information Processing Systems* 33:1428–1439.

Dai, W.-Z.; Xu, Q.; Yu, Y.; and Zhou, Z.-H. 2019. Bridging machine learning and logical reasoning by abductive learning. *Advances in Neural Information Processing Systems* 32.

Daniele, A., and Serafini, L. 2019. Knowledge enhanced neural networks. In *Proceedings of the 16th Pacific Rim International Conference on Artificial Intelligence (PRICAI)*, Lecture Notes in Computer Science, 542–554. Springer.

Daniele, A., and van Krieken, E. 2026. Gradient-based optimization on gödel logic as discrete local search. *arXiv preprint arXiv:2503.01817*.

Daniele, A.; Campari, T.; Malhotra, S.; and Serafini, L. 2022. Deep symbolic learning: Discovering symbols and rules from perceptions. *arXiv preprint arXiv:2208.11561*.

Deng, L.; Jiao, P.; Pei, J.; Wu, Z.; and Li, G. 2018. Gxnor-net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework. *Neural Networks* 100:49–58.

Dierckx, L.; Veroneze, R.; and Nijssen, S. 2023. Rl-net: Interpretable rule learning with neural networks. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 95–107. Springer.

Diligenti, M.; Gori, M.; and Sacca, C. 2017. Semantic-based regularization for learning and inference. *Artificial Intelligence* 244:143–165.

Dougherty, J.; Kohavi, R.; and Sahami, M. 1997. Supervised and unsupervised discretization of continuous features. *ICML* 1995.

Evans, R., and Grefenstette, E. 2018. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research* 61:1–64.

Gao, K.; Inoue, K.; Cao, Y.; Wang, H.; and Feng, Y. 2025. Differentiable rule induction from raw sequence inputs. In *The Thirteenth International Conference on Learning Representations*.

Gumbel, E. J. 1954. *Statistical theory of extreme values and some practical applications: a series of lectures*, volume 33. US Government Printing Office.

Huang, Y.-X.; Dai, W.-Z.; Cai, L.-W.; Muggleton, S. H.; and Jiang, Y. 2021. Fast abductive learning by similarity-based consistency optimization. *Advances in Neural Information Processing Systems* 34:26574–26584.

Jang, J.-S. 1993. Anfis: adaptive-network-based fuzzy inference system. *IEEE transactions on systems, man, and cybernetics* 23(3):665–685.

Katzir, L.; Elidan, G.; and El-Yaniv, R. 2020. Net-dnf: Effective deep modeling of tabular data. In *International conference on learning representations*.

Kingma, D. P. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kusters, R.; Kim, Y.; Collery, M.; Marie, C. d. S.; and Gupta, S. 2022. Differentiable rule induction with learned relational features. *arXiv preprint arXiv:2201.06515*.

Manhaeve, R.; Dumancic, S.; Kimmig, A.; Demeester, T.; and De Raedt, L. 2018. Deepproblog: Neural probabilistic logic programming. *Advances in neural information processing systems* 31.

Marra, G.; Dumančić, S.; Manhaeve, R.; and De Raedt, L. 2024. From statistical relational to neurosymbolic artificial intelligence: A survey. *Artificial Intelligence* 328:104062.

Moro, S.; Rita, P.; and Cortez, P. 2014. Bank Marketing. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5K306.

Nguyen, N. N.; Zhou, W. J.; and Quek, C. 2015. Gsetsk: a generic self-evolving tsk fuzzy neural network with a novel hebbian-based rule reduction approach. *Applied Soft Computing* 35:29–42.

Payani, A., and Fekri, F. 2019. Inductive logic programming via differentiable deep neural logic networks. *arXiv preprint arXiv:1906.03523*.

Perreault, V.; Inoue, K.; Labib, R.; and Hertz, A. 2025. Neural logic networks for interpretable classification. *arXiv preprint arXiv:2508.08172*.

Petersen, F.; Borgelt, C.; Kuehne, H.; and Deussen, O. 2022. Deep differentiable logic gate networks. *Advances in Neural Information Processing Systems* 35:2006–2018.

Petersen, F.; Kuehne, H.; Borgelt, C.; Welzel, J.; and Ermon, S. 2024. Convolutional differentiable logic gate networks. *Advances in Neural Information Processing Systems* 37:121185–121203.

Qiao, L.; Wang, W.; and Lin, B. 2021. Learning accurate and interpretable decision rule sets from neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 4303–4311.

Riegel, R.; Gray, A.; Luus, F.; Khan, N.; Makondo, N.; Akhalwaya, I. Y.; Qian, H.; Fagin, R.; Barahona, F.; Sharma, U.; et al. 2020. Logical neural networks. *CoRR* abs/2006.13155.

Rocktäschel, T., and Riedel, S. 2017. End-to-end differentiable proving. *Advances in neural information processing systems* 30.

Shihabudheen, K., and Pillai, G. 2018. Recent advances in neuro-fuzzy system: A survey. *Knowledge-Based Systems* 152:136–162.

Topan, S.; Rolnick, D.; and Si, X. 2021. Techniques for symbol grounding with satnet. *Advances in Neural Information Processing Systems* 34:20733–20744.

Tsamoura, E.; Hospedales, T.; and Michael, L. 2021. Neural-symbolic integration: A compositional perspective. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, 5051–5060.

Wang, P.-W.; Donti, P.; Wilder, B.; and Kolter, Z. 2019. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, 6545–6554.

Wang, Z.; Zhang, W.; Wang, J.; et al. 2020. Transparent classification with multilayer logical perceptrons and random binarization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, 6331–6339.

Winters, T.; Marra, G.; Manhaeve, R.; and De Raedt, L. 2022. Deepstochlog: Neural stochastic logic programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 10090–10100.

Xu, J.; Zhang, Z.; Friedman, T.; Liang, Y.; and Van den Broeck, G. 2018. A semantic loss function for deep learning with symbolic knowledge. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, 5502–5511. PMLR.

Yang, Z.; Ishay, A.; and Lee, J. 2020. Neurasp: Embracing neural networks into answer set programming. In *29th International Joint Conference on Artificial Intelligence, IJCAI 2020*, 1755–1762. International Joint Conferences on Artificial Intelligence.

Yousefi, S.; Plesner, A.; Aczel, T.; and Wattenhofer, R. 2025. Mind the gap: Removing the discretization gap in differentiable logic gate networks. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Zhou, R., and Quek, C. 1996. A pseudo outer-product based fuzzy neural network and its rule-identification algorithm. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, 1156–1161 vol.2.