TU Delft

Neuro-Symbolic AI

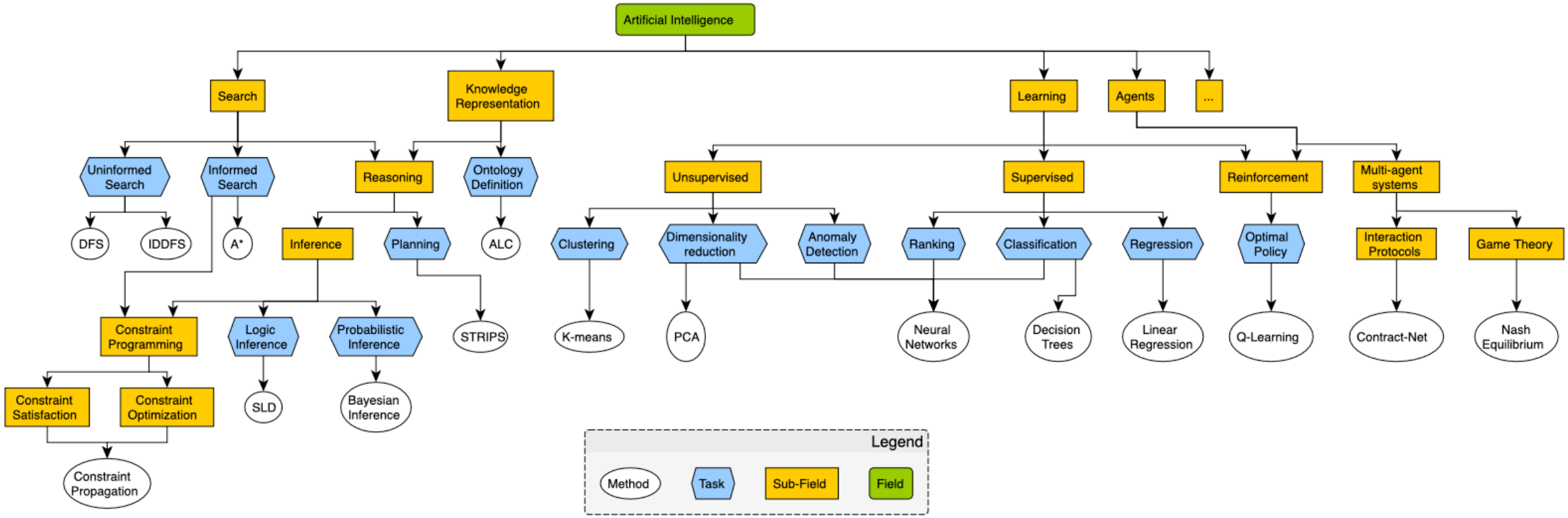
The What and Why

Andrea Agiollo

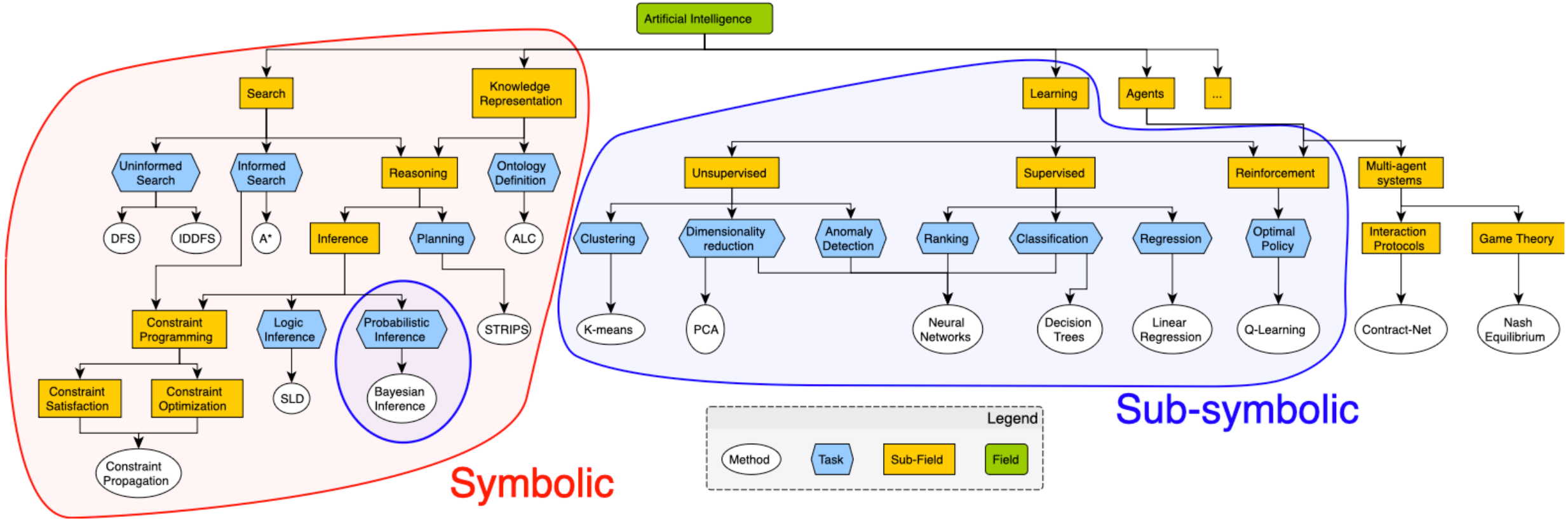
15-11-2026



Quick and Dirty Overview of AI



Symbolic and Sub-Symbolic AI



Examples of Symbolic AI

- Logic programming: sentences in logical form, representing domain knowledge. Computation performed applying logical reasoning to that knowledge, to solve problems in the domain.
  - Prolog, Answer Set Programming, Datalog
- Knowledge representation: model information in structured manner to formally represent it as knowledge (incorporates psychology findings about how humans represent knowledge)
  - Semantic Web (e.g., OWL), Description Logics (e.g., ALC)
- Automated reasoning: simulate human reasoning processes to solve problems or derive conclusions from known facts
  - Theorem proving, Model checking
- Planning in intelligent agents: automatic generation of sequences of actions that achieve specific goals given model of the environment and constraints
  - Stanford Research Institute Problem Solver, PDDL

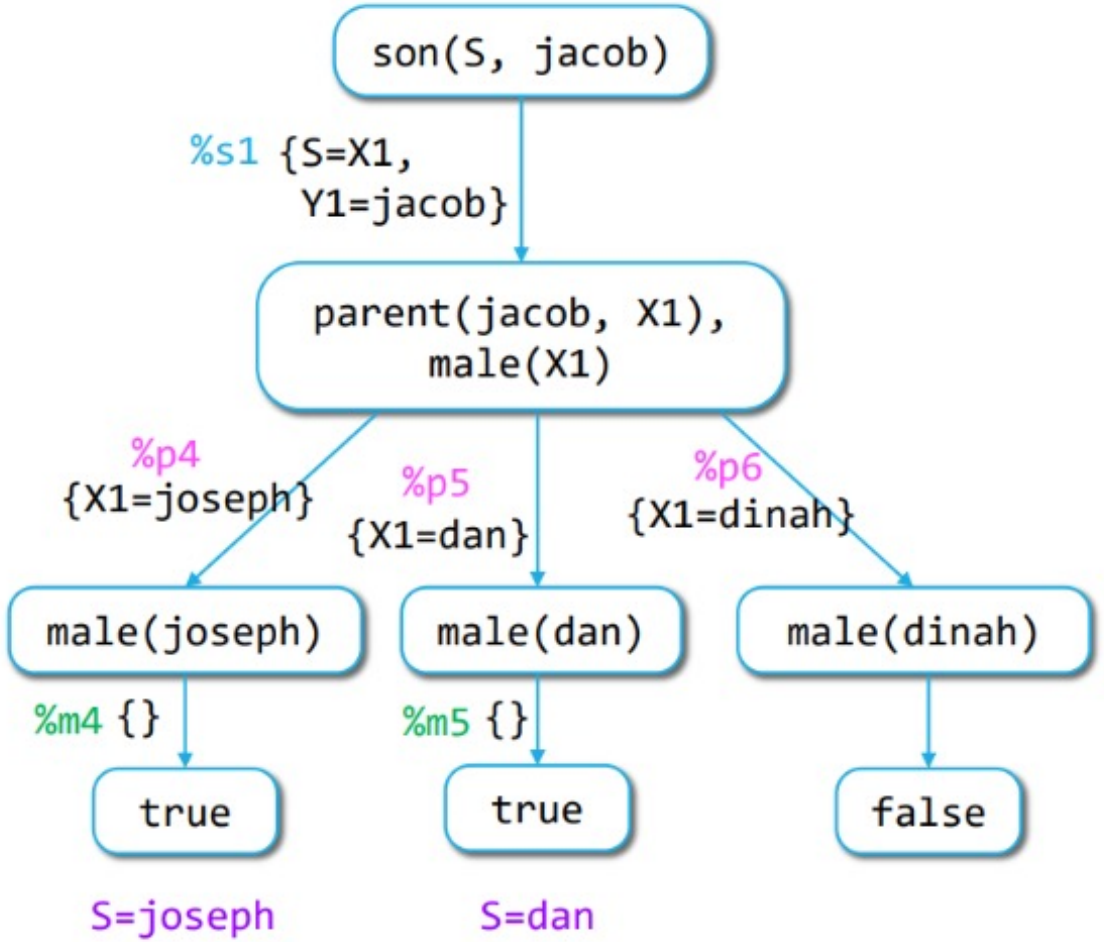
Logic Programming

```
parent(abraham, isaac). %p1
parent(isaac, jacob).   %p2
parent(sarah, isaac).   %p3
parent(jacob, joseph).  %p4
parent(jacob, dan).     %p5
parent(jacob, dinah).   %p6

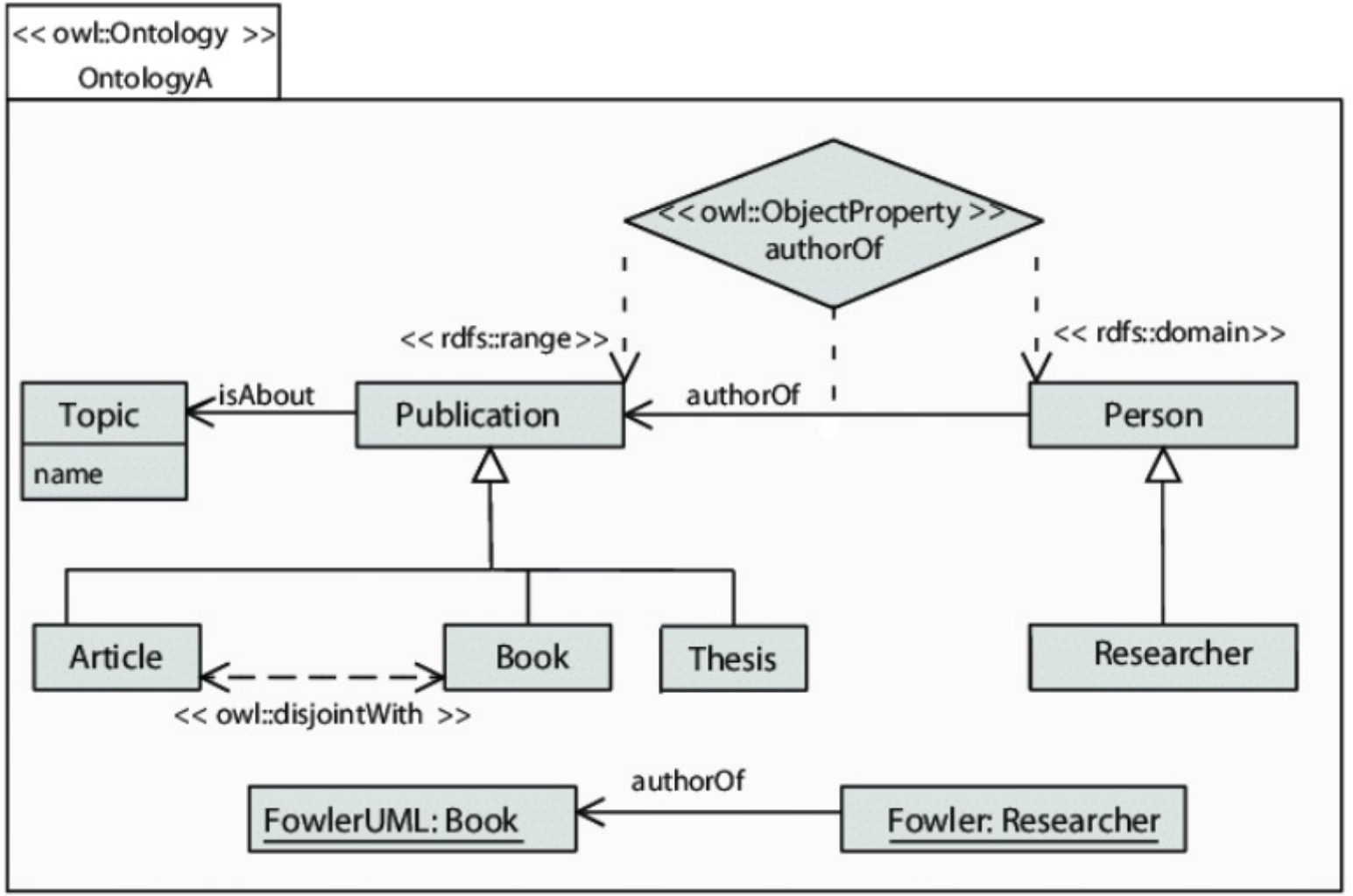
male(abraham).          %m1
male(isaac).            %m2
male(jacob).            %m3
male(joseph).           %m4
male(dan).              %m5

son(X,Y) :- parent(Y,X), %s1
            male(X).
```

```
?- son(S,jacob).
```



Knowledge Representation



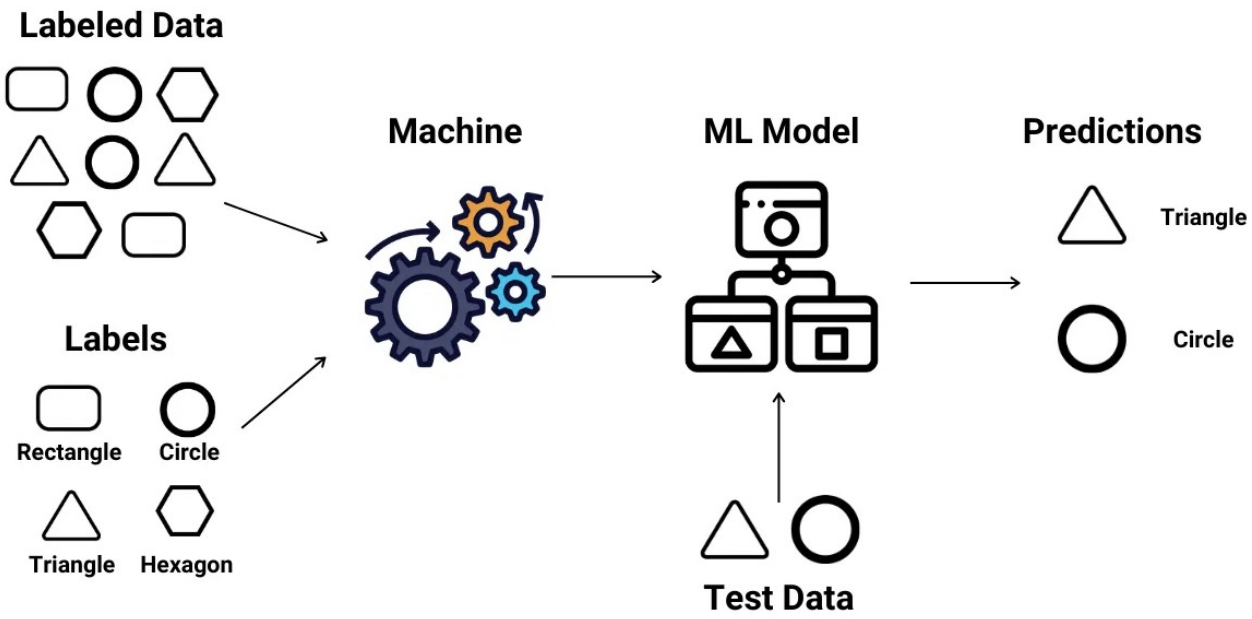
Common Characteristics of Symbolic AI

- Structured representations: knowledge (I/O data) is represented in a structured, formal way (e.g., logic formulas, ontologies)
- Algorithmic manipulation of representations: each approach relies on algorithms that manipulate these structured representations following exact rules
- Crisp semantics: the meaning of the representations is well-defined, and the algorithms produce exact results
  - representations are either well-formed or not, algorithms rely on rules which are either applicable or not
- Model-driven: algorithms may commonly work in zero- or few-shot settings, humans must commonly model and encode knowledge in the target structure
- Clear computational complexity: the decidability, complexity, and tractability of the algorithms are well understood

Examples of Sub-Symbolic AI

- Machine learning: algorithms that can learn from data, generalise to unseen data and perform tasks without explicit instructions
  - Supervised learning: fitting a discrete (classification) or a continuous function (regression) from examples
  - Unsupervised learning: clustering, dimensionality reduction
  - Reinforcement learning: learning a policy to maximize a reward signal, via simulation
- Probabilistic reasoning: extends traditional logic with probabilistic expressions to account for to logical entailment and lack of evidence
  - Bayesian networks, Markov models, probabilistic logic programming

Supervised Learning



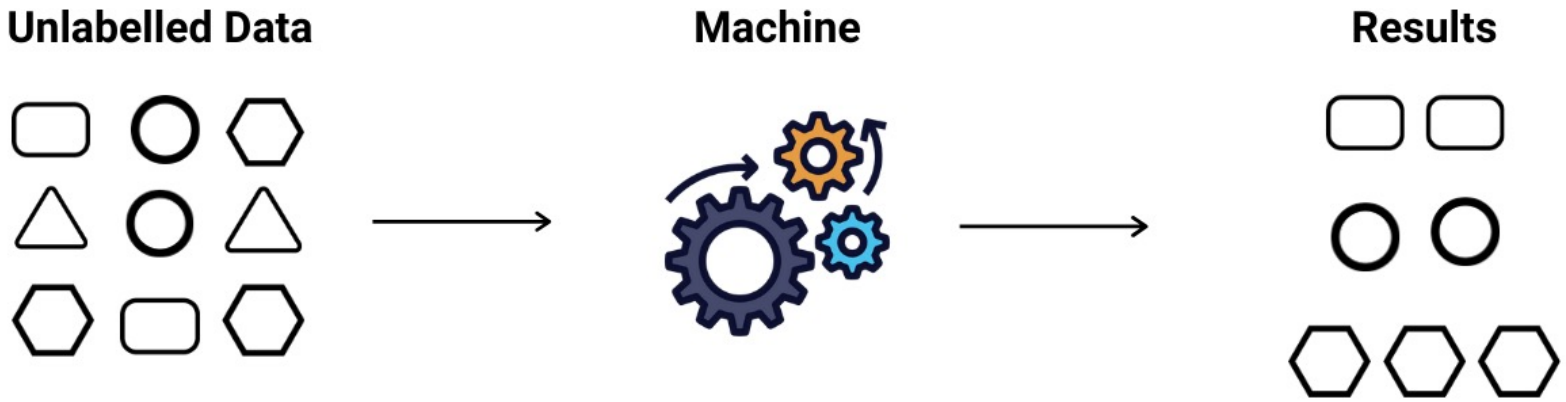
DailyDoseofDS.com

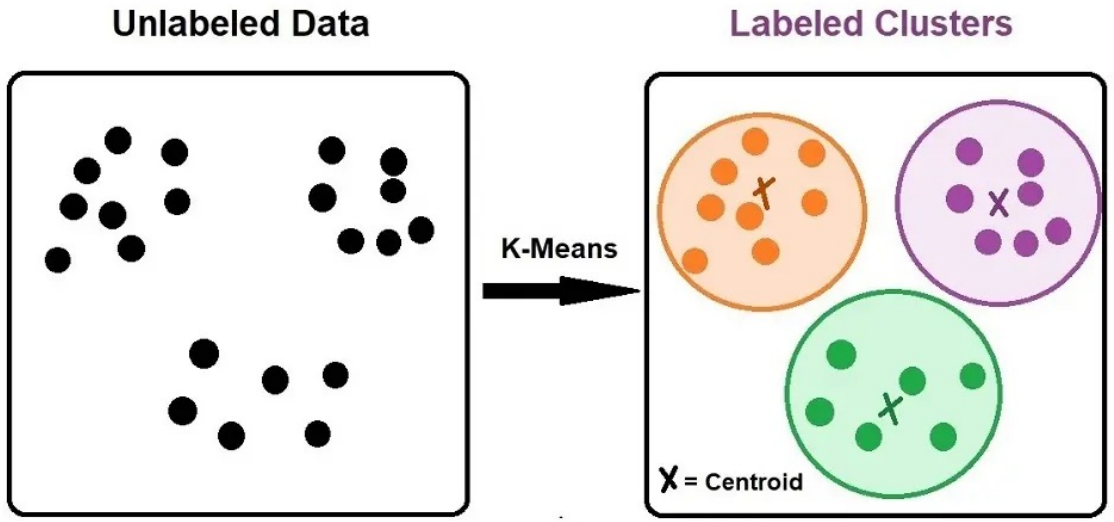
10 Most Common Loss Functions in Machine Learning

by Avi Chawla

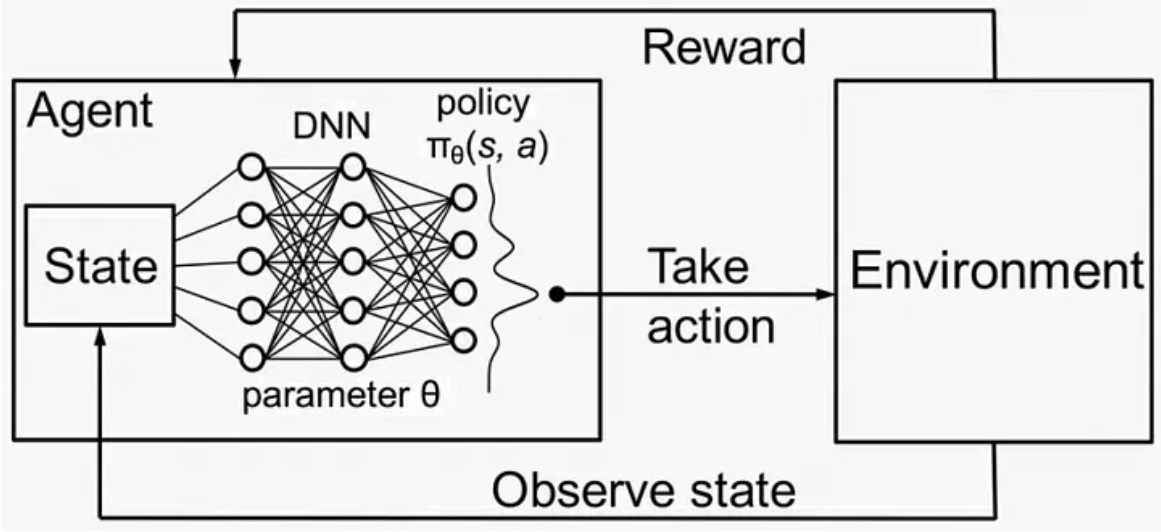
Loss Function Name	Description	Function
Regression Losses		
Mean Bias Error	Captures average bias in prediction. But is rarely used for training.	$\mathcal{L}_{MBE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))$
Mean Absolute Error	Measures absolute average bias in prediction. Also called L1 Loss.	$\mathcal{L}_{MAE} = \frac{1}{N}\sum_{i=1}^{N}\lvert y_i - f(x_i)\rvert$
Mean Squared Error	Average squared distance between actual and predicted. Also called L2 Loss.	$\mathcal{L}_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2$
Root Mean Squared Error	Square root of MSE. Loss and dependent variable have same units.	$\mathcal{L}_{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2}$
Huber Loss	A combination of MSE and MAE. It is parametric loss function.	$\mathcal{L}_{\text{Huberloss}} = \begin{cases} \frac{1}{2}(y_i - f(x_i))^2 & : \lvert y_i - f(x_i)\rvert \le \delta \\ \delta(\lvert y_i - f(x_i)\rvert - \frac{1}{2}\delta) & : otherwise \end{cases}$
Log Cosh Loss	Similar to Huber Loss + non-parametric. But computationally expensive.	$\mathcal{L}_{\text{LogCosh}} = \frac{1}{N}\sum_{i=1}^{N} log(cosh(f(x_i) - y_i))$
Classification Losses (Binary + Multi-class)		
Binary Cross Entropy (BCE)	Loss function for binary classification tasks.	$\mathcal{L}_{BCE} = \frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(x_i)) + (1 - y_i) \cdot log(1 - p(x_i))$
Hinge Loss	Penalizes wrong and right (but less confident) predictions. Commonly used in SVMs.	$\mathcal{L}_{\text{Hinge}} = max(0, 1 - (f(x) \cdot y))$
Cross Entropy Loss	Extension of BCE loss to multi-class classification.	$\mathcal{L}_{CE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} . log(f(x_{ij}))$ $N : samples; M : classes$
KL Divergence	Minimizes the divergence between predicted and true probability distribution	$\mathcal{L}_{KL} = \sum_{i=1}^{N} y_i \cdot log(\frac{y_i}{f(x_i)})$

Unsupervised Learning





Reinforcement Learning



$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

Old Q Value
Reward
Discount Rate
(0 ~ 1)
New Q Value
Learning Rate
(0 ~ 1)
Maximum Q value of
transition destination state
TD error

Common Characteristics of Sub-Symbolic AI

- Numeric representations: knowledge (I/O data) is represented in a less structured way, often as vectors/matrices/tensors of numbers
- Differentiable manipulation of representations: algorithms rely on mathematical operations involving these numeric representations, most-commonly undergoing some optimization process
- Fuzzy/continuous semantics: representations are from continuous spaces, where similarities and distances are defined in a continuous way, and algorithms may yield fuzzy results
- Data-driven + Usage vs. training: algorithms are often trained on data, to be later re-used on other data. Usage is commonly impractical or impossible without training
- Computational complexity: strong reliance on greedy or time-limited optimization methods,
- Lack of theoretical guarantees on generalizability: complex to check whether obtained model work on unseen data or not

Long Standing Dualism

Symbolic AI

Advantages:

- Transparent reasoning
- Verifiable
- No training data required

Disadvantages:

- Knowledge acquisition bottleneck
- Missing flexibility
- Not applicable to large unstructured information

Sub-Symbolic AI

Advantages:

- Adaptability
- Effective learning from raw data
- Robustness to noisy/unstructured information

Disadvantages:

- Opaqueness/interpretability
- Lack of guarantees on generalizability
- Data hungriness
- Security issues

Long Standing Dualism

- Intuition vs. Reasoning:
  - Esprit de finesse vs. Esprit de géométrie (Philosophy): Blaise Pascal, 1669
  - Cognitive vs. Behavioural Psychology: B.F. Skinner, 1950s
  - System 1 (fast, intuitive) vs. System 2 (slow, rational): Daniel Kahneman, 2011
- Sub-symbolic AI ideal for system 1: fast, intuitive, patten-matching intelligence
- Symbolic AI ideal for system 2: slow, rational, verifiable and complete logic reasoning

Neuro-Symbolic AI

- Symbolic and Sub-Symbolic approaches are complementary
- Advantages of one cover disadvantages of the other
- Idea: integrate/combine symbolic and sub-symbolic approaches together
- Rough definition: *Neuro-symbolic AI is a hybrid AI paradigm that integrates symbolic reasoning (logic, rules, knowledge graphs) with neural learning (deep learning, pattern recognition) to enable systems that can both learn from data and reason about it.*
- Neuro-symbolic AI aims:
  - Learn from raw data like images or text (neural networks)
  - Reason about abstract concepts and relationships (symbolic logic)
  - Explain decisions in human-understandable terms

Neuro-Symbolic AI Taxonomy

General purpose taxonomy proposed by Kautz et al:

- Symbolic Neural symbolic: symbols converted into numerical representations processed by NNs whose output is converted back into symbols.
  - LLMs, words converted into vectors processed by NNs (SPOILER: I do not agree)
- Symbolic[Neural] symbolic techniques used to invoke neural network sub-components.
  - AlphaGo: Monte Carlo tree search + NN to evaluate game positions
- Neural | Symbolic: neural architecture interprets perceptual data as symbols and relationships that are reasoned about symbolically.
  - Neural-Concept Learner: object-based scene representation + translates sentences into executable symbolic programs
- Neural: Symbolic → Neural: symbolic reasoning to generate or label data used by a NN
- NeuralSymbolic: NN generated from symbolic rules.
  - Neural Theorem Prover constructs a NN from an AND-OR proof tree generated from knowledge base rules and terms
- Neural[Symbolic]: NN calls symbolic reasoning engine to perform an action.
  - ChatGPT using a plugin to query Wolfram Alpha (SPOILER: I do not agree)

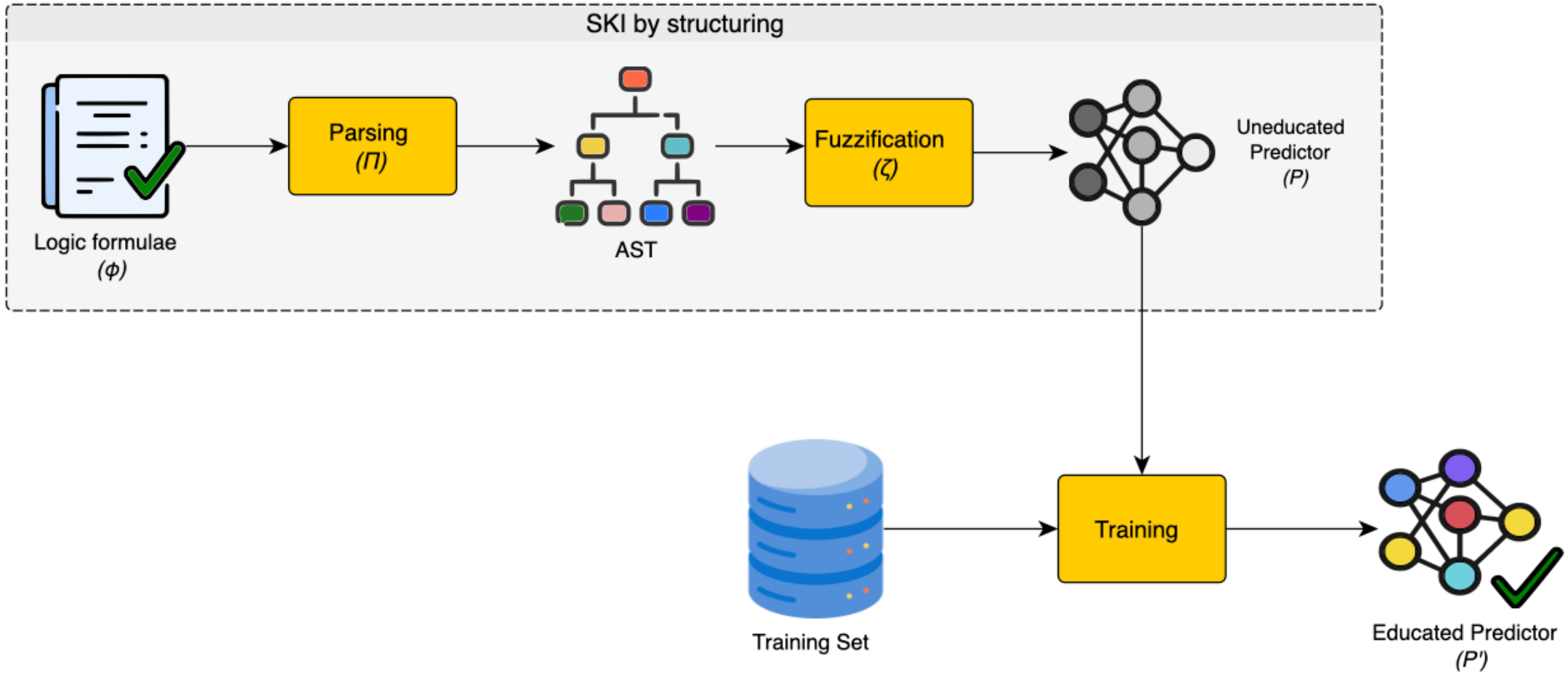
Neuro-Symbolic AI Taxonomy

- Kautz et al taxonomy is general purpose
- Focus on any combination of symbolic and sub-symbolic techniques
- Some classes do not pro-actively integrate symbolic and sub-symbolic (e.g., Symbolic Neural symbolic and Neural[Symbolic])
- Focus on actual integration of symbolic and sub-symbolic components (i.e., use symbolic to modify how sub-symbolic computes or vice-versa)
- Important to focus on *trustworthiness* and *robustness* in machine learning
- Our focus: Symbolic Knowledge Injection (SKI) defined as injecting symbolic knowledge into sub-symbolic models
- SKI definition: *Any algorithmic procedure affecting how sub-symbolic predictors draw their inferences in such a way that predictions are either computed as a function of, or made consistent with, some given symbolic knowledge.*

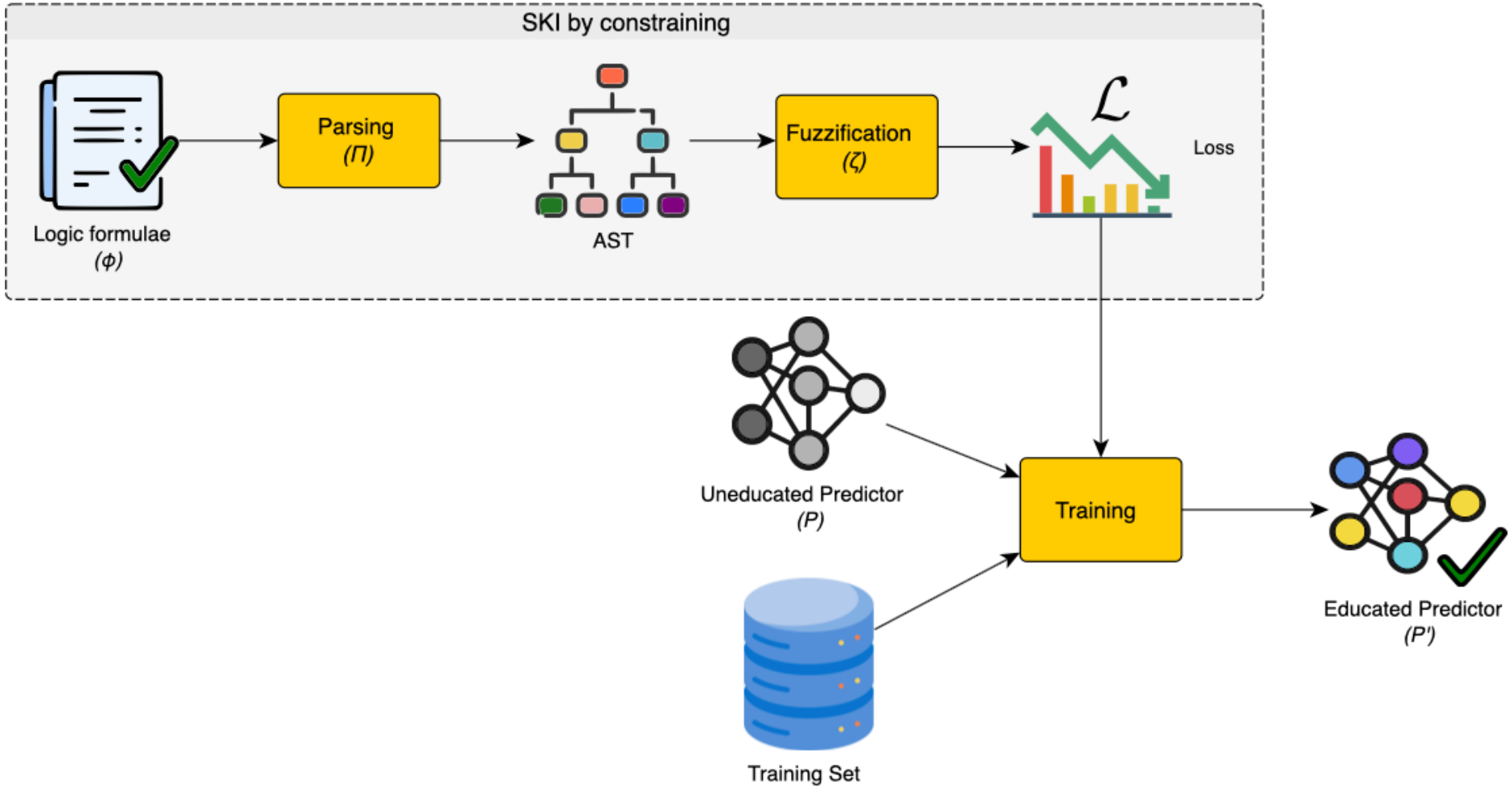
New SKI Taxonomy

- Ciatto et al novel taxonomy focusing specifically on SKI
- How does SKI work?
  - Predictor structuring: (a part of) a sub-symbolic predictor is created to mirror the symbolic knowledge via its own internal structure. NNs whose internal structure is crafted to represent logic predicates via neurons, and logic connectives via synapses.
  - Guided learning: symbolic knowledge is used to steer the learning process of ML predictors by either penalizing inconsistent behaviors or by incentivizing consistent behaviors about the symbolic knowledge. Achieved by altering the loss function such that violations of the SK increase the loss.
  - Knowledge embedding: symbolic knowledge converted into numeric-array form to be provided as ‘ordinary’ input for the sub-symbolic predictor undergoing injection (it is my paper but I don’t agree lol)

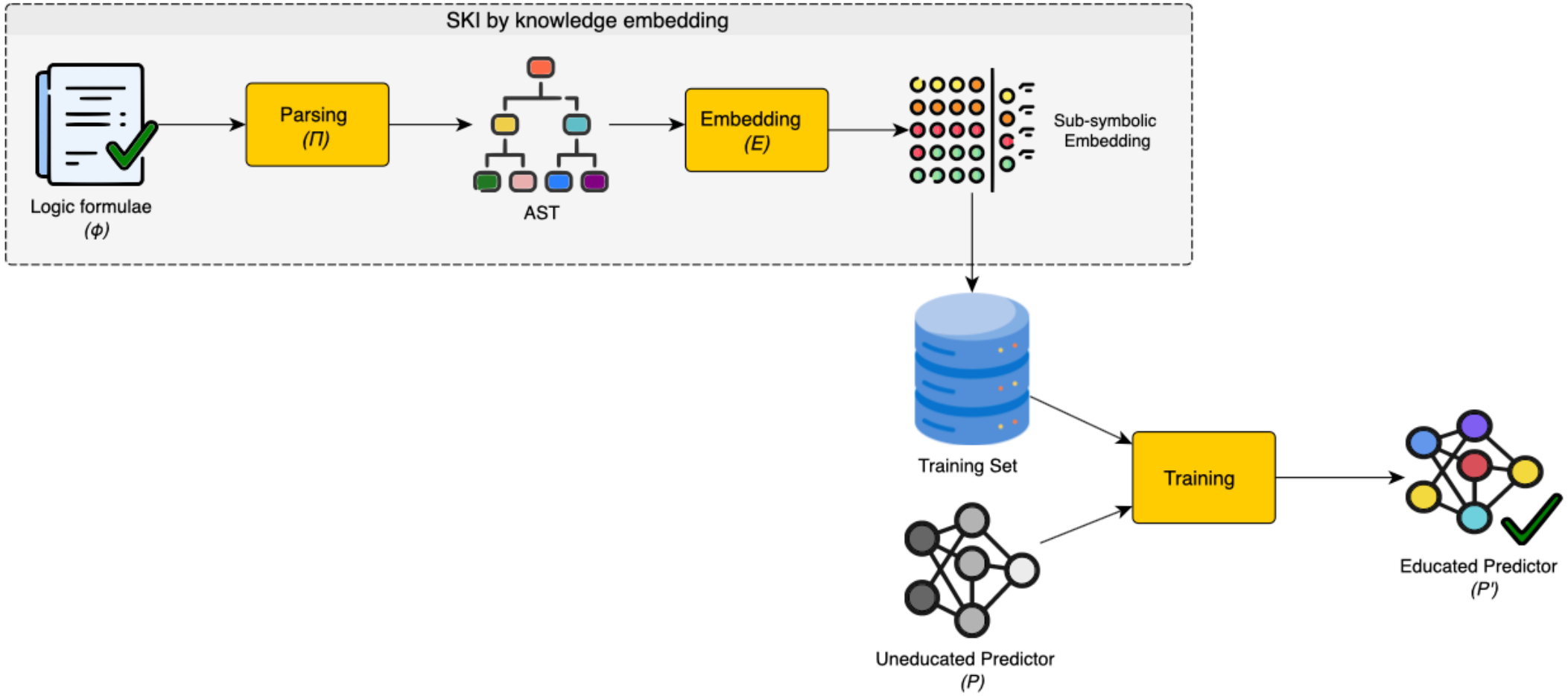
SKI via Structuring



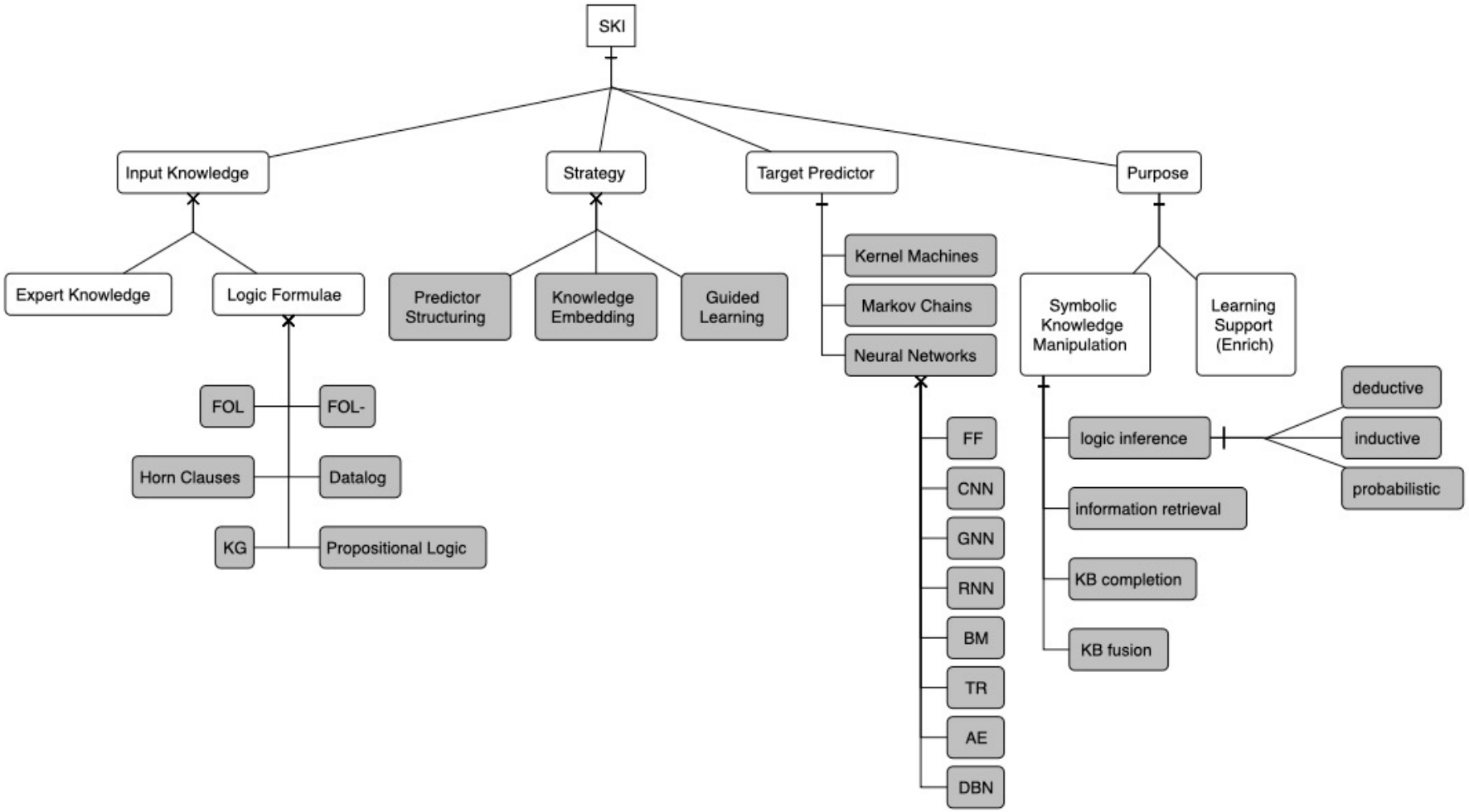
SKI via Guided Learning



SKI via Embedding



Complete SKI Taxonomy



NeSy/SKI Example: DeepProbLog

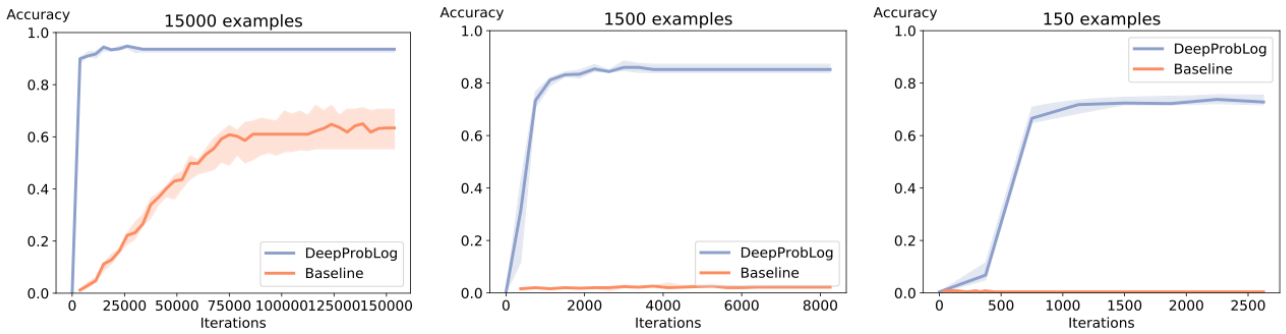
- DeepProbLog = ProbLog + Neural Predicates
- Intuition: use NN to estimate the probability of some predicates or facts in a ProbLog program
- NN normalized output interpreted as probabilities that become probabilistic fact
- NN components are instantiated as (neural) annotated disjunction
  - e.g., nn(mnist_net, [X], Y, [0 ... 9]) :: digit(X,Y).
- NN weights are optimized together with ProbLog probabilities end-to-end using gradient descent on seen samples

NeSy/SKI Example: DeepProbLog

- Toy example: digits addition, given two handwritten digits the model should return the sum of the two.
- Use NN predicate to recognize the digits and write logic to compute the sum:

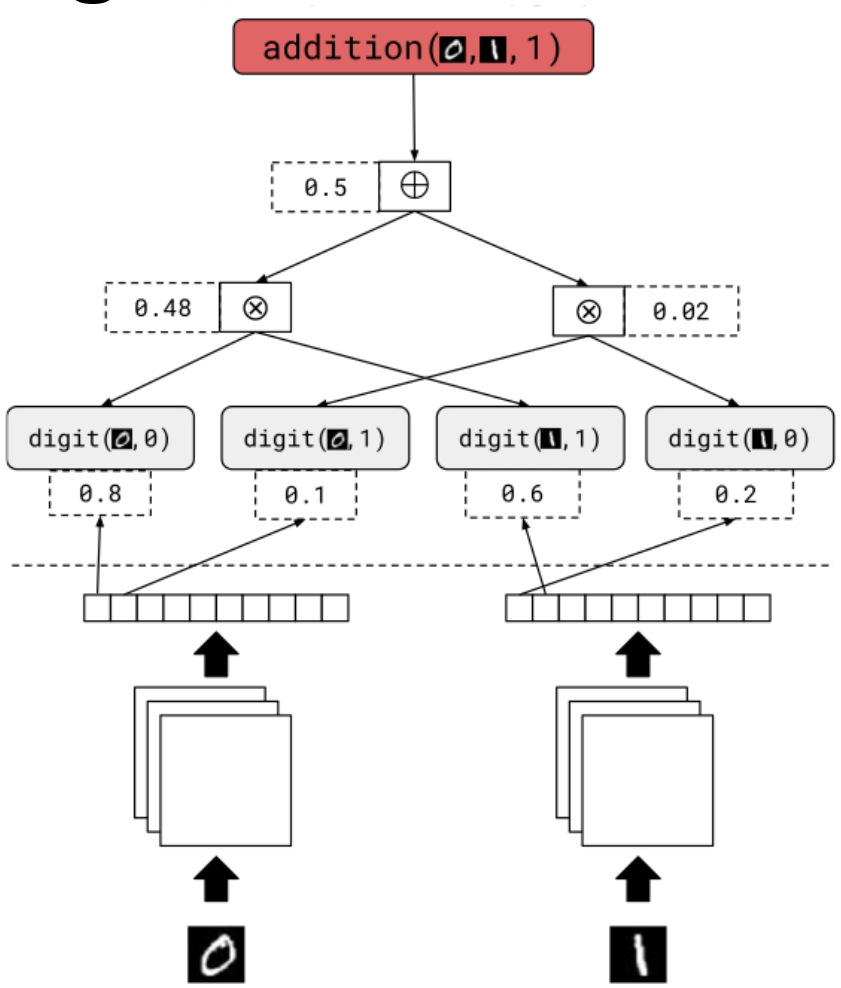
```
nn(m_digit, [X], Y, [0...9]) :: digit(X,Y).
addition(X,Y,Z) :- digit(X,N1), digit(Y,N2), Z is N1+N2.
```

- Learns with much fewer examples w.r.t. to NNs:



(a) 15 000 examples (b) 1 500 examples (c) 150 examples

Fig. 7. MNIST Multi-Digit Addition (**T2**). The graphs show the accuracy on the validation set during training for different training set sizes.



(d) The AC for query addition(0, 1, 1).

NeSy/SKI Example: DeepProbLog

```
nn(classifier, [X], Y, [0 .. 9]) :: digit(X,Y).
t(0.2) :: noisy.

1/19 :: uniform(X,Y,0) ; ... ; 1/19 :: uniform(X,Y,18).

addition(X,Y,Z) :- noisy, uniform(X,Y,Z).
addition(X,Y,Z) :- \+noisy, digit(X,N1), digit(Y,N2), Z is N1+N2.
```

```
nn(net1, [X], Y, [heads, tails]) :: coin1(X,Y).
nn(net2, [X], Y, [heads, tails]) :: coin2(X,Y).

compare(X,X,same).
compare(X,Y,different) :- \+compare(X,Y,same).

coins(X,Comparison) :-
    coin1(X,C1),
    coin2(X,C2),
    compare(C1,C2,Comparison).
```

```
t(1/4)::house_rank(jack);t(1/4)::house_rank(queen);
    t(1/4)::house_rank(king);t(1/4)::house_rank(ace).
nn(net1,[X],Y,[jack,queen,king,ace]):: rank(X,Y).

hand(Cards,straight(low)) :-
    member(card(jack),Cards),
    member(card(queen),Cards),
    member(card(king),Cards).
hand(Cards,straight(high)) :-
    member(card(queen),Cards),
    member(card(king),Cards),
    member(card(ace),Cards).
hand([card(R), card(R), card(R)],threeofakind(R)).
hand(Cards,pair(R)) :-
    select(card(R),Cards,Cards2),
    member(card(R),Cards2).
hand(Cards,high(R)) :-
    member(card(R),Cards).

hand_rank(high(jack),0).
...
hand_rank(straight(high),13).

best_hand_rank(Cards,R) :-
    hand(Cards,H),
    hand_rank(H,R),
    \+(hand(Cards,H2),hand_rank(H2,R2),R2>R).

outcome(R1,R2,win) :- R1 > R2.
outcome(R1,R2,loss) :- R1 < R2.
outcome(R,R,draw).

cards(C1,C2,House,[card(R1), card(R2), House]) :-
    rank(C1,R1),
    rank(C2,R2).

game([C1,C2,C3,C4],House,Outcome) :-
    cards(C1,C2,House,Hand1),
    cards(C3,C4,House,Hand2),
    best_hand_rank(C1,R1),
    best_hand_rank(C2,R2),
    outcome(R1,R2,Outcome).

game(Cards,Outcome) :-
    house_rank(House),
    game(Cards,House,Outcome).
```

```
:-use_module(library(apply)).

nn(net1,[T],R,[child,child_in_law,...]) :: nn_rel(T,R).
nn(net2,[T],G,[male,female]) :: nn_gender(T,G).

query_rel(T,X,R,Y):-gender_rel(R2,G,R),neural_gender(T,Y,G),call(R2,T,X,Y,3).

mentioned(X,s(E,_)) :- member(X,E).

neural_gender([Entities|Sentences], E, G) :-
    member(E,Entities),
    include(mentioned(E), Sentences, [H|Sentences2]),
    maplist(embed([E]),[H|Sentences2],Embedded),
    max_tensor(Embedded,Max),
    nn_gender(Max,G).

neural([Entities|Sentences], X,R,Y) :-
    member(X,Entities), member(Y,Entities), X \== Y,
    include(mentioned(X),Sentences,Sentences2),
    include(mentioned(Y),Sentences2,[H|Sentences3]),
    maplist(embed([Y,X]),[H|Sentences3],Embedded),
    max_tensor(Embedded,Max),
    nn_rel(Max,R).


child(T,X,Y,_) :- neural(T,X,child,Y).
child_in_law(T,X,Y,_) :- neural(T,X,child_in_law,Y).
grandchild(T,X,Y,_) :-  neural(T,X,grandchild,Y).
...

grandchild(T,X,Y,D) :- D>0,X\={Y}, child(T,X,Z,D-1), child(T,Z,Y,D-1).
grandchild(T,X,Y,D) :- D>0,X\={Y}, so(T,X,Z,D-1), grandchild(T,Z,Y,D-1).
grandchild(T,X,Y,D) :- D>0,X\={Y}, grandchild(T,X,Z,D-1), sibling(T,Z,Y,D-1).
grandparent(T,X,Y,D) :- D>0,X\={Y}, parent(T,X,Z,D-1), parent(T,Z,Y,D-1).
grandparent(T,X,Y,D) :- D>0,X\={Y}, sibling(T,X,Z,D-1), grandparent(T,Z,Y,D-1).
child(T,X,Y,D) :- D>0,X\={Y}, child(T,X,Z,D-1), sibling(T,Z,Y,D-1).
child(T,X,Y,D) :- D>0,X\={Y}, so(T,X,Z,D-1), child(T,Z,Y,D-1).
parent(T,X,Y,D) :- D>0,X\={Y}, sibling(T,X,Z,D-1), parent(T,Z,Y,D-1).
parent(T,X,Y,D) :- D>0,X\={Y}, child(T,X,Z,D-1), grandparent(T,Z,Y,D-1).
sibling(T,X,Y,D) :- D>0,X\={Y}, child(T,X,Z,D-1), uncle(T,Z,Y,D-1).
sibling(T,X,Y,D) :- D>0,X\={Y}, parent(T,X,Z,D-1), child(T,Z,Y,D-1).
sibling(T,X,Y,D) :- D>0,X\={Y}, sibling(T,X,Z,D-1), sibling(T,Z,Y,D-1).
child_in_law(T,X,Y,D) :- D>0,X\={Y}, child(T,X,Z,D-1),so(T,Z,Y,D-1).
parent_in_law(T,X,Y,D) :- D>0,X\={Y}, so(T,X,Z,D-1), parent(T,Z,Y,D-1).
nephew(T,X,Y,D) :-D>0,X\={Y}, sibling(T,X,Z,D-1), child(T,Z,Y,D-1).
uncle(T,X,Y,D) :- D>0,X\={Y}, parent(T,X,Z,D-1), sibling(T,Z,Y,D-1)

gender_rel(child,male,son).
gender_rel(child,female,daughter).
gender_rel(parent,male,father).
gender_rel(parent,female,mother).
```

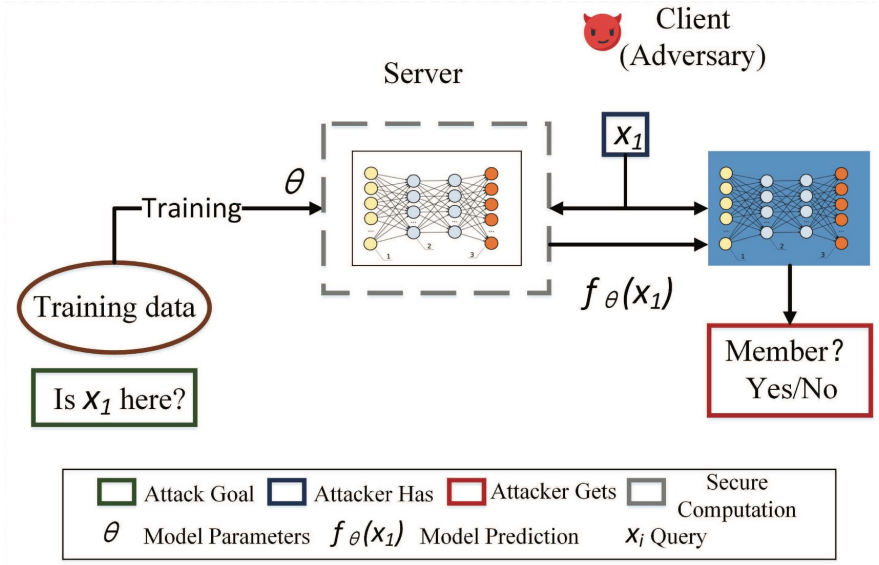
NeSy/SKI Example: SemanticLoss

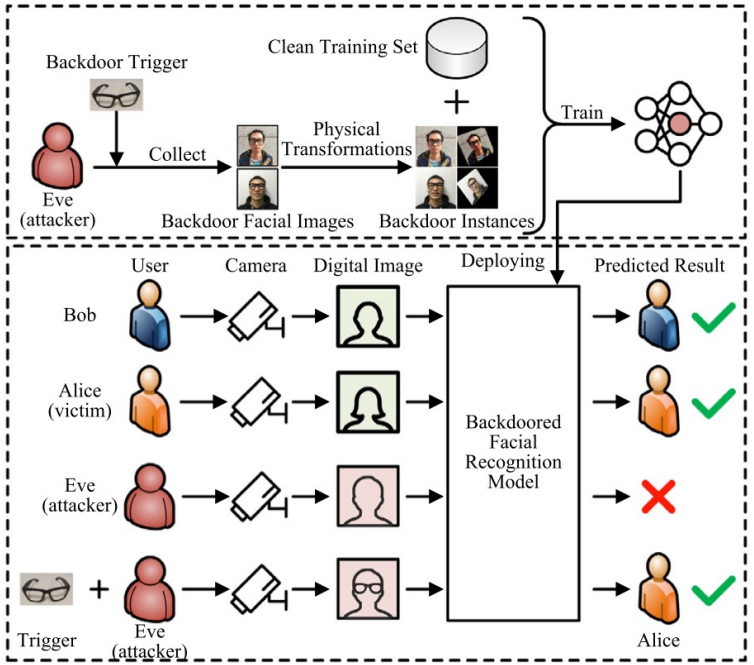
- Define a loss function that allow the model to satisfy a set of invariant logic properties about its output
- Starting from a sentence α in predicate logic, defined over some atoms X = $\{X_1, ..., X_n\}$
- Each atom X_i is associated with a probability p_i as the output of a neural model
- Let the vector $\mathbf{p}$ denote the collection of probabilities $p_1, ..., p_n$ over the atoms X
- Relevant properties to consider:
  - If $\alpha \models \beta$, then $L(\alpha, p) \geq L(\beta, p)$ for any value of p.
  - If vector x satisfies the constraint α it should have no loss
  - $L(X, p) \propto -\log(p)$ and $L(\neg X, p) \propto -\log(1-p)$

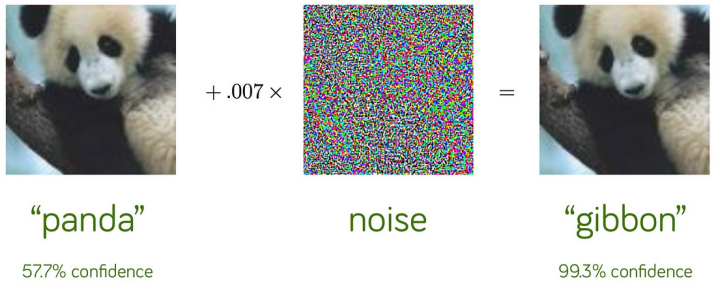
$$\mathrm{L}^{\mathrm{s}}(\alpha, \mathbf{p}) \propto -\log \sum_{\mathbf{x} \models \alpha} \prod_{i:\mathbf{x} \models X_i} \mathbf{p}_i \prod_{i:\mathbf{x} \models \neg X_i} (1 - \mathbf{p}_i).$$

Why Should CYSes Care About NeSy?

- Neural networks are susceptible to a large set of security issues:
  - Adversarial attacks
  - Backdoor attacks
  - Membership inference attacks
  - Model inversion attacks







Why Should CYSes Care About NeSy?

- The verifiable, transparent and robust symbolic components of NeSy approaches make them secure by design, right?

Why Should CYSes Care About NeSy?

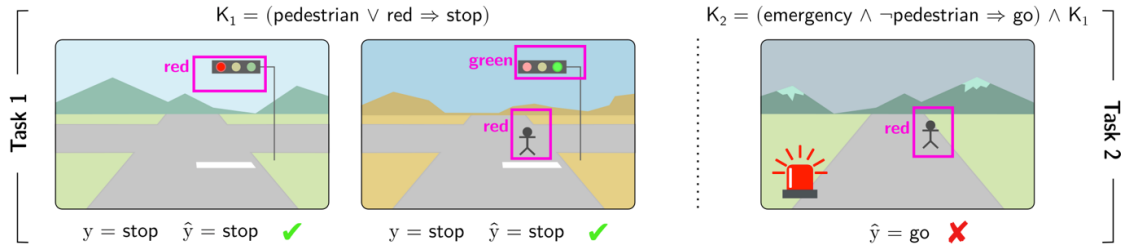
- The verifiable, transparent and robust symbolic components of NeSy approaches make them secure by design, right?



Why Should CYSes Care About NeSy?

- Guided learning NeSy approaches (e.g., SemanticLoss) do not provide guarantee that symbolic knowledge is concretely learned by the model
- Structuring NeSy approaches (e.g., DeepProbLog) suffer reasoning shortcuts:
  - Unintended strategies learned by NeSy models that allow to satisfy symbolic constraints or make correct predictions without truly understanding underlying concepts
  - Often exploit spurious correlations in the data and lead to poor generalization and misleading reasoning

TASK	EXAMPLE DATA	KNOWLEDGE K	EXAMPLE RS	IMPACT
MNMath	$\begin{cases} 2 \cdot \blacksquare + \blacksquare = 6 \\ \blacksquare + \blacksquare = 7 \end{cases}$	Equations must hold.	$\begin{cases} \blacksquare \to 2 \\ \blacksquare \to 4 \\ \blacksquare \to 3 \end{cases}$	$\blacksquare + \blacksquare = 5$



Why Should CYSes Care About NeSy?

- Ongoing project: *Reasoning Shortcuts activation using Backdoor attacks*
- Manipulate training data to define custom reasoning shortcuts in NeSy models
- Activate specific reasoning shortcuts at deployment phase
- Several challenging tasks:
  - How to optimally choose data to be manipulated to target a specific RS?
  - How to identify which reasoning shortcuts exist when attacker does not know the NeSy model?
  - How to select which reasoning shortcuts are easier to activate (from an attacker perspective) if multiple shortcuts attain the same effect?

Why Should CYSes Care About NeSy?

- Not everything is bad
- NeSy can be used as a tool to help defense systems
- It provides more transparent, manageable and robust decision support systems
- Possible application scenario on sensitive systems where ML is getting increasingly popular:
  - Intrusion detection systems: Use symbolic knowledge of threats and assets to constrain ML-based detection engines -> symbolic constraints help avoid flagging known safe behaviors as threats (false positive reduction), rapid response to emerging threats with insertion of new rules, fixing concept/data drift issues
  - Malware analysis: NN analyze binary code or behavior patterns while symbolic reasoning interprets these patterns in the context of known malware families -> Identify new malware variant linking it to a known threat actor's tactics
  - Automated incident response: symbolic system encode response playbooks, while neural components mine high-level concept patterns from logs and alerts -> automatically isolate compromised endpoint based on inferred threat severity and policy rules
  - Phishing detection: neural component mines suspicious email content, URLs, and sender while symbolic system applies rules to assess domain reputation, email header anomalies, and policy violations -> refined alerts
  - Vulnerability prioritization: assess severity based on CVE descriptions and/or exploit patterns using NNs and cross-references with asset criticality and business impact rules using symbolic reasoner -> Unknown vulnerability can be categorized and prioritized for patching if it affects a critical server and has a known exploit in the wild

Why Should CYSes Care About NeSy?

- Ongoing project: *Enhancing ML-based IDS using Attack Graph and Vulnerability Knowledge*
- Use symbolic knowledge of threats and assets to constrain ML-based detection engines
- Symbolic knowledge include characteristics of attack graphs:
  - Attack path existence
  - Connection between specific vulnerability and attack
  - Hierarchical relations in attack types
  - Suggestions?
- Use SKI to inject knowledge into detection module while still training on traffic data

Questions?