

Towards Neurosymbolic Learning for the Semantic Web with Kolibrie

Volodymyr Kadzhaia*, Pieter Bonte

Department of Computer Science, KU Leuven Campus Kulak, Belgium

Abstract

Neuro-symbolic systems combine statistical perception with explicit logical knowledge, but their integration with Semantic Web (SW) infrastructure remains incomplete. Neural predictions are usually external scores, while rules, ontologies, provenance, and explanations live in RDF stores and SPARQL query engines. This paper argues that neuro-symbolic learning can be made native to the SW by treating uncertain facts as first-class RDF statements and by making their probabilities and provenance accessible through SPARQL-star. Probabilistic circuits are an orthogonal reasoning component: they are used to compute probabilities over symbolic derivations, while RDF-star provides the Semantic Web representation layer. We present a compact architecture, implemented in Kolibrie, where RDF-star annotations represent weighted base and derived triples, rules define structured tasks and constraints, and SPARQL-star makes uncertainty, provenance, and training signals queryable.

Keywords

Neuro-symbolic learning, Semantic Web, RDF 1.2, SPARQL-star

1. Introduction

The Semantic Web (SW) offers a mature stack for querying and reasoning over structured knowledge. RDF provides graph-shaped data, SPARQL provides a query language, and rule or ontology layers provide deductive structure. Modern AI systems, however, often enter this stack only as external services: a neural model predicts a class, emits a confidence score, and the graph stores the result as an ordinary literal. This separation loses much of what is needed for neuro-symbolic learning: the score is not part of the logical semantics, alternative proofs are not tracked, correlations between derivations are ignored, and explanations are difficult to query.

Neural-symbolic reasoning has a long history of connecting learned representations with logical structure [1]. Systems such as DeepProbLog show that neural predictions can be treated as probabilistic choices inside a logic program [1]. Our question is how to bring the same idea into SW practice, where the native interfaces are RDF graphs, SPARQL queries, Linked Data identifiers, and provenance-aware graph annotations.

We propose to make the uncertainty produced by neural models part of the SW data model itself. The central idea is simple: if a graph triple is a claim, then a weighted triple is a claim with probabilistic semantics. RDF-star provides a natural representation for such weights, because it allows statements about statements without replacing the RDF graph by a separate probabilistic data structure [2]. For example, a neural relation can assert that a sensor is likely to show high risk, while RDF-star records its probability:

```
<< ex:sensor17 ex:riskSignal ex:high >>
  prob:value "0.83"^^xsd:double .
```

Neuro-symbolic reasoning also needs a way to compute probabilities for rule-derived conclusions. Weighted model counting (WMC) is a standard route for probabilistic logic programs [3]. Several representations can be used for this purpose. In this paper we focus on Sentential Decision Diagrams

Posters, Demos, Blue Sky, and Tutorials at SEMANTICS 2026, Sep 2026, Ghent, Belgium

*Corresponding author.

✉ volodymyr.kadzhaia@kuleuven.be (V. Kadzhaia); pieter.bonte@kuleuven.be (P. Bonte)

id 0009-0000-1715-7782 (V. Kadzhaia); 0000-0002-8931-8343 (P. Bonte)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

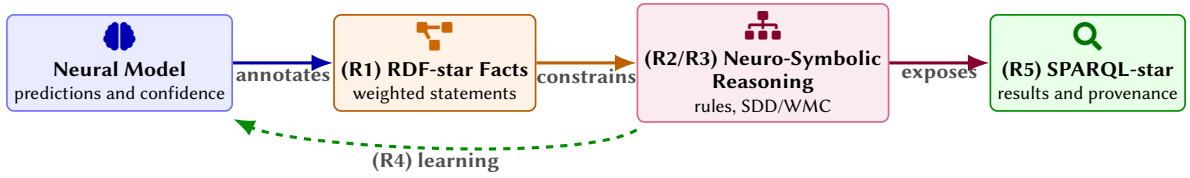


Figure 1: Compact Kolibrie pipeline with the requirements mapped onto the workflow: weighted RDF-star evidence (R1), symbolic and probabilistic reasoning (R2–R3), learning feedback (R4), and queryable provenance (R5).

(SDDs) because they can represent more expressive Boolean constraints, such as mutually exclusive neural outputs and cardinality restrictions that are typical within OWL2 ontologies [4].

This paper sketches how these mechanisms bring neuro-symbolic learning to SW systems. We use Kolibrie [5], a Rust-based RDF, SPARQL, and stream-reasoning engine, as the implementation setting. Prior Kolibrie work already allowed pretrained models to be used inside the SW stack; here we focus on making training part of the same RDF/SPARQL-oriented workflow. The contribution is not a new neural architecture, but a semantic integration pattern: (i) RDF-star stores weighted facts and explanations, (ii) rules define symbolic constraints and derived targets, (iii) probabilistic circuits compute derivation probabilities, and (iv) SPARQL-star exposes the resulting probabilities, proofs, and features for downstream learning.

2. Requirements for Neuro-Symbolic Learning

A SW system becomes neuro-symbolic only when learning and reasoning share semantics, rather than merely exchanging files. We identify five requirements:

- R1 Weighted evidence as graph data.** Neural outputs, sensor reliabilities, user feedback, or weak labels must be represented as graph facts with weights. The representation should preserve the identity of the underlying statement, because later rules need to refer to both the fact and its confidence.
- R2 Symbolic composition.** Domain knowledge should be expressible as rules over graph patterns: joins, filters, negation, and ontology-derived relationships. The symbolic layer defines how uncertain evidence composes into higher-level concepts such as alarms, recommendations, diagnoses, or policy violations.
- R3 Expressive probabilistic reasoning.** Alternative derivations of the same conclusion may share evidence, so a system must track proof structure rather than only scalar confidence. For simple rule fragments, semiring-style provenance may be sufficient; more expressive SW constraints, such as mutually exclusive choices or cardinality restrictions, require a representation that can compile and count Boolean proof constraints.
- R4 Differentiable feedback.** Learning requires a path from symbolic loss back to neural parameters. When neural outputs are represented as weighted random variables, the gradient of a target probability with respect to those weights can be used as feedback for the neural model. The symbolic program can therefore act as a structured loss function.
- R5 Queryable provenance.** Neuro-symbolic systems need explanations for debugging, trust, and training data curation. The probabilities, proof counts, and contributing seeds should remain queryable through SW mechanisms instead of being hidden in an external inference engine.

3. RDF-star as the Weight Layer

RDF-star gives a compact notation for annotating a triple while keeping the triple itself addressable, addressing **R1** and **R5** in Fig. 1. Kolibrie uses quoted triples as internal terms and exposes their annotations through SPARQL-star. Kolibrie keeps the asserted triple and its weight synchronized

internally: the triple is stored as graph data, while its probability is represented as an RDF-star annotation of that same statement. For example, the internal representation can be exposed as:

```
ex:sensor17 ex:tempEvidence true .
<< ex:sensor17 ex:tempEvidence true >>
  prob:value "0.72"^^xsd:double .
```

The same representation is used for derived facts. After rule evaluation, Kolibrie materializes a probability annotation for each derived triple and can additionally materialize explanation triples such as proof counts, proof identifiers, and links to the weighted seed triples. This is important for interoperability: a downstream SPARQL query does not need to know whether a probability came from a neural classifier, a manually provided reliability value, or an SDD-based inference result.

This design also keeps the SW contract intact. Weighted annotations are metadata about graph statements, while the asserted graph can still be queried by ordinary SPARQL patterns. Applications that do not use uncertainty can ignore the RDF-star layer; applications that do use uncertainty can query it directly:

```
1 SELECT ?sensor ?p ?proofCount WHERE {
2   << ?sensor ex:riskSignal ex:high >> prob:value ?p .
3   << ?sensor ex:riskSignal ex:high >> prob:proofCount ?proofCount .
4 }
```

RDF-star is therefore not only a serialization convenience. In this architecture it is the boundary object between neural prediction, symbolic reasoning, explanation, and learning.

4. SDD-Based Model Counting

Kolibrie’s provenance interface treats each fact as carrying a tag: an implementation-level annotation whose type and operations are defined by a provenance semiring, following the semiring-annotation view of database provenance [6]. Classical reasoning uses Boolean tags, following the Boolean-semiring view of classical set semantics in provenance frameworks [6, 7]; approximate probabilistic reasoning can use min–max or add–multiply tags [7]; exact probabilistic reasoning uses proof formulas. The SDD provenance backend represents each input weight as a variable and each derived fact as an SDD handle. Rule conjunction is compiled as Boolean conjunction, alternative derivations as disjunction, and stratified negation as circuit negation.

Kolibrie rules are non-standard extensions inspired by SPARQL CONSTRUCT: the WHERE part matches graph patterns and the CONSTRUCT part materializes derived triples. The rule syntax selects the desired provenance semantics, addressing **R2** and **R3**:

```
1 RULE :RiskFromSignals PROB(combination=sdd) :-
2 CONSTRUCT { ?s ex:riskSignal ex:high . }
3 WHERE { ?s ex:tempEvidence true . ?s ex:pressureEvidence true . }
```

For independent evidence, each variable x_i receives positive weight p_i and negative weight $1 - p_i$. For neural classifiers with exclusive outputs, SDDs can encode an “exactly one” constraint over the class variables, so that the probability mass of a prediction distribution is not treated as a set of independent labels, echoing the role of annotated disjunctions in probabilistic logic programming [8, 1]. This matters for learning: the symbolic layer can ask for probabilities of classes, disjunctions of classes, or rule-defined concepts without breaking the classifier’s output semantics.

5. Making Learning Native in Kolibrie

Kolibrie exposes learning through special declarations. MODEL declares the neural model, NEURAL RELATION states which RDF graph patterns provide its input features, and TRAIN NEURAL RELATION connects those features to labelled graph data. This is the training part of **R4** in Fig. 1.

The **MODEL** component declares the network and its output space. **OUTPUT EXCLUSIVE** lists all mutually exclusive labels; for a binary predicate, **OUTPUT BINARY { ex:high }** can name only the positive label and leave the negative case implicit.

```
1 MODEL "risk_model" {
2   ARCH MLP { HIDDEN [16, 8] }
3   OUTPUT EXCLUSIVE { ex:low, ex:high }
4 }
```

NEURAL RELATION binds the model to RDF input patterns. Here, **?sensor** is the entity being classified, while **?t** and **?p** are the numeric feature variables passed to the neural model.

```
1 NEURAL RELATION ex:riskSignal USING MODEL "risk_model" {
2   INPUT { ?sensor ex:temp ?t . ?sensor ex:pressure ?p . }
3   FEATURES { ?t, ?p }
4 }
```

TRAIN NEURAL RELATION supplies labelled examples. The **DATA** pattern retrieves training labels from triples such as **sensor17 ex:goldRisk ex:high**; **TARGET** states that the model is trained to produce **?sensor ex:riskSignal ?label**.

```
1 TRAIN NEURAL RELATION ex:riskSignal {
2   DATA { ?sensor ex:goldRisk ?label . }
3   LABEL ?label
4   TARGET { ?sensor ex:riskSignal ?label }
5   LOSS cross_entropy
6   OPTIMIZER adam
7   LEARNING_RATE 0.001
8   EPOCHS 80
9   BATCH_SIZE 16
10  SAVE_TO "risk_model.bin"
11 }
```

After training, Kolibrie-specific **ML.PREDICT** can be used inside a rule to call the saved model over a SPARQL input query and materialize predictions as RDF triples. The **CONSTRUCT** clause states the triple to be created, while **OUTPUT** binds the predicted object value.

```
1 RULE :PredictRisk :-
2 CONSTRUCT { ?sensor ex:riskSignal ?risk . }
3 WHERE { ?sensor ex:observed true . }
4 ML.PREDICT(MODEL "risk_model",
5   INPUT {
6     SELECT ?sensor ?t ?p
7     WHERE { ?sensor ex:temp ?t . ?sensor ex:pressure ?p . }
8   },
9   OUTPUT ?risk
10 )
```

A useful consequence is that the learned predicate can later participate in ordinary or probabilistic rules; this is not required after every prediction, but it enables the symbolic layer to reason over ML outputs.

```
1 RULE :HighRiskAlert PROB(combination=sdd) :-
2 CONSTRUCT { ?sensor ex:alert ex:high . }
3 WHERE { ?sensor ex:riskSignal ex:high . ?sensor ex:pressureEvidence true . }
```

Thus the learning syntax defines how graph data become neural input, while probabilistic rules define how learned predicates are constrained, explained, and counted through RDF-star and SDD provenance.

6. Open Research Questions

Several research questions remain. (i) SDD-backed WMC is expressive enough for constraints beyond simple Datalog-shaped rules, but compactness depends on how the circuit is compiled. Compared with explicit proof enumeration, SDDs provide a compact circuit representation on which WMC can be computed after compilation. (ii) Streaming data introduces incrementality challenges: window updates may change the proof structure, so future work should study how to maintain probabilistic circuits without recomputing them from scratch. (iii) RDF-star needs a clear vocabulary for probabilities, proof identity, and model metadata.

Despite these questions, the architecture is deliberately incremental. Existing RDF data remain valid, SPARQL-star queries can inspect weighted annotations, and applications can choose between Boolean, approximate, and SDD-backed provenance. This makes the system suitable for gradual adoption in knowledge-graph applications where not every fact is uncertain and not every query requires exact model counting.

7. Conclusion

Bringing neuro-symbolic learning to the SW requires more than calling neural models from a graph pipeline. It requires a shared semantics for uncertain evidence, symbolic derivations, probabilities, and explanations. RDF-star provides a graph-native way to identify and annotate weighted statements. SDDs provide an exact and reusable representation for model counting over correlated proofs. Together, they allow Kolibrie to expose neural predictions, rule-derived facts, WMC probabilities, and provenance through the same SW interface. This positions RDF-star and SDD-backed provenance as practical building blocks for neuro-symbolic SW systems.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT assistant for language editing. The authors reviewed the content and take full responsibility for it.

References

- [1] R. Manhaeve, S. Dumancic, A. Kimmig, T. Demeester, L. De Raedt, Deepproblog: Neural probabilistic logic programming, *Advances in neural information processing systems* 31 (2018).
- [2] O. Hartig, P.-A. Champin, G. Kellogg, A. Seaborne, RDF-star and SPARQL-star, Technical Report, W3C Community Group Final Report, 2021. URL: <https://www.w3.org/2021/12/rdf-star.html>.
- [3] D. Fierens, G. Van den Broeck, J. Renkens, D. Shterionov, B. Gutmann, I. Thon, G. Janssens, L. De Raedt, Inference and learning in probabilistic logic programs using weighted boolean formulas, *Theory and Practice of Logic Programming* 15 (2015) 358–401.
- [4] A. Darwiche, Sdd: A new canonical representation of propositional knowledge bases, in: *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, 2011, p. 819.
- [5] V. Kadzhaia, P. Bonte, Neuro-symbolic stream reasoning with kolibrie, in: M. Acosta, M. van Erp, S. Rudolph, O. Hartig, B. Spahiu, A. Rula, D. Garijo, F. Osborne (Eds.), *The Semantic Web*, Springer Nature Switzerland, Cham, 2026, pp. 39–57.
- [6] T.J. Green, G. Karvounarakis, V. Tannen, Provenance semirings, in: *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2007, pp. 31–40.
- [7] Z. Li, J. Huang, M. Naik, Scallop: A language for neurosymbolic programming, *Proceedings of the ACM on Programming Languages* 7 (2023) 1463–1487.
- [8] J. Vennekens, S. Verbaeten, M. Bruynooghe, Logic programs with annotated disjunctions, in: *International Conference on Logic Programming*, Springer, 2004, pp. 431–445.