

# DeepLog: A Software Framework for Modular Neurosymbolic AI

Robin Manhaeve, Stefano Colamonaco, Vincent Derkinderen,  
Rik Adriaensen, Lucas Van Praet, Luc De Raedt, Giuseppe Marra

Department of Computer Science and Leuven.AI  
KU Leuven, Belgium  
{firstname.lastname}@kuleuven.be

## Abstract

*DeepLog* is an operational neurosymbolic framework that unifies logic and deep learning within standard PyTorch workflows. While existing neurosymbolic systems focus on a particular paradigm and semantics, *DeepLog* serves as a universal backend that can emulate many systems in the neurosymbolic “alphabet soup”. By treating diverse neurosymbolic languages as high-level specifications, the *DeepLog* software automatically compiles them into optimized arithmetic circuits. This design lowers the barrier for machine learning practitioners by treating logic as composable modules, while providing neurosymbolic developers with a shared, high-performance basis for prototyping new integration strategies. The code is available here: <https://github.com/ML-KULeuven/deeplog>

## 1 Introduction

Neurosymbolic AI (NeSy) aims to combine deep learning with symbolic reasoning and representations Hitzler and Sarker (2022); Garcez and Lamb (2023); Marra et al. (2024). However, the current landscape resembles an “alphabet soup” of seemingly disconnected paradigms and frameworks. Several dimensions are commonly used to distinguish among these approaches, including the nature of their semantics (e.g., fuzzy or probabilistic), the integration paradigm (e.g., incorporation of symbolic representations into the loss function or the architecture of the machine learning model), and the type of symbolic front-end employed (e.g., first-order logic or logic programming). Because these differences are often treated as fundamental rather than alternative instantiations, new approaches are generally pursued within non-interoperable frameworks. As a result, no cross-paradigm software tool has yet emerged, slowing systematic comparison and cumulative progress in the field.

To address this gap, we contribute and demo the *DeepLog* software that provides building blocks and primitives for neurosymbolic AI, enabling abstraction over commonly used representations and computational mechanisms. It is based on the *DeepLog* abstract neurosymbolic machine defined in Derkinderen et al. (2025), and can be used to easily implement, reconstruct and compare a wide range of neurosymbolic systems. To this end, it consolidates essential components from the existing “alphabet soup” of approaches and offers the necessary primitives for designing new ones.

The *DeepLog* software exploits modern deep learning soft-

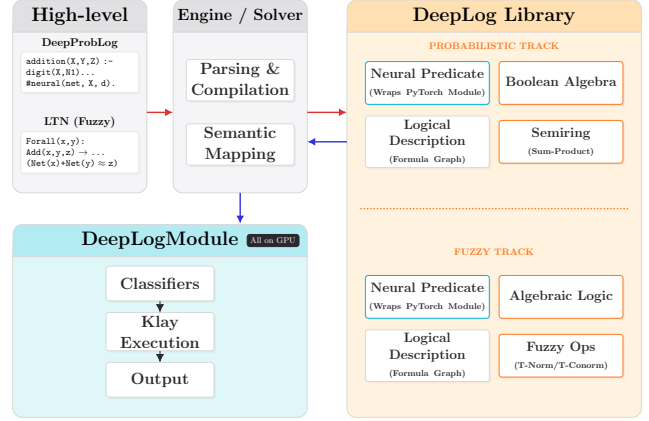


Figure 1: The High-Level Specification (top-left) allows users to switch between semantics simply by changing the underlying logic track in the *DeepLog* Library (right). The system automatically compiles these specifications into an optimized *DeepLogModule* (bottom-left) for execution on the GPU.

and hardware. First, it extends standard PyTorch modules with symbolic annotations on their inputs and outputs, based on formulas expressed in the *DeepLog* language Derkinderen et al. (2025), enabling the software to manage composition and integration automatically. Second, it does not only use GPU-acceleration for neural modules, but also for its symbolic components. Most other neurosymbolic AI systems implement their symbolic component on the CPU, which not only degrades performance due to data transfer and synchronization, but also introduces additional complications for debugging, and potential errors in terms of what data is on what device.

While most existing NeSy systems adopt a monolithic design encompassing model specification, data handling, and training, *DeepLog* is designed to be modular and non-intrusive. Rather than replacing established machine learning workflows, it can be integrated selectively into existing pipelines. This design choice lowers the barrier to entry for machine learning practitioners interested in neurosymbolic AI, while simultaneously providing framework developers with a flexible basis for prototyping new ideas without sacrificing performance.

**The main contribution** of this work is an implementation of *DeepLog*, an abstract neurosymbolic machine, that

- allows us to reconstruct any member of the alphabet soup with minimal code,

- provides out-of-the-box GPU acceleration,
- provides modularity as automatic module compositions,
- integrates with modern ML software (PyTorch).

## 2 Bridging Paradigms

The integration of neurosymbolic systems remains largely an open challenge, with only a few approaches explicitly aiming at unification. Most of them focus on the uniform encoding of multiple types of semantics, such as LYRICS Marra et al. (2019) and Pylon Ahmed et al. (2022). However, these frameworks are usually limited to a specific neurosymbolic paradigm, most often semantic-based regularization, where logical constraints are added as loss terms during training. In addition, they rely on a single front-end representation, which limits their use across the broader neurosymbolic landscape. More recent work, such as ULLER van Krieken et al. (2024), attempts to overcome this limitation by introducing a unified language that can express a wide range of neurosymbolic specifications. However, this unification mainly takes place at the high-level front-end layer. At present, no public implementation is available, so its practical usefulness is still unclear. Moreover, because the unification is restricted to the front-end, it has a strong impact on user interaction, as different research communities prioritize different syntactic structures and modeling abstractions.

## 3 DeepLog for ML Practitioners

This section introduces DeepLog from the perspective of machine learning practitioners who wish to integrate symbolic reasoning into existing PyTorch-based workflows. The central idea is that symbolic information attached to module interfaces enables automatic composition, validation, and transformation of neurosymbolic components, while remaining compatible with standard deep learning practices.

### 3.1 Module Composition

Neurosymbolic models are typically constructed by composing heterogeneous components, such as neural network predictors, logical functions or constraints. Each of these components assumes that its inputs represent specific semantic quantities, for example Boolean values, probabilities, or continuous fuzzy scores. In practice, these assumptions are rarely made explicit, leading to fragile compositions and error-prone glue code. DeepLog addresses this through `DeepLogModules`, which are standard PyTorch modules augmented with symbolic annotations on their inputs and outputs. These annotations specify the semantic role of each tensor and are represented using (tuples of) `SymTensors`: N-dimensional tensors whose entries are associated with symbolic expressions.

These symbolic interfaces allow DeepLog to automatically compose modules. When chaining two modules, DeepLog automatically derives and inserts the necessary GPU-accelerated transformations between them. The annotations also enable run-time validation of module interfaces,

ensuring that inputs and outputs satisfy the expected symbolic structure, significantly improving debuggability.

DeepLog supports two primary composition patterns. First, modules can be composed sequentially, analogous to `torch.nn.Sequential`, where the output of one module feeds directly into the next. Second, when a collection of modules forms a directed acyclic graph based on their symbolic input–output dependencies, DeepLog can automatically wire the full computation graph, including all required intermediate transformations. By leveraging PyTorch’s asynchronous execution model, this approach maximizes parallelism and GPU utilization.

**Example 1** (Semantic loss from a DIMACS constraint). *We demonstrate semantic loss Xu et al. (2018) using a logical constraint loaded from a DIMACS CNF specification. Assume a classifier produces probabilities for classes  $\{A, B, C\}$ , and domain knowledge enforces a hierarchy:  $A \Rightarrow B$  and  $C \Rightarrow B$ . This constraint is encoded once as a Boolean CNF and reused independently of the neural model.*

```
dimacs = """
p cnf 3 2
-1 2 0 c A -> B
-3 2 0 c C -> B
"""

phi = parse_dimacs_to_module(dimacs,
                             structure='probability')
expected = SymTensor(["A_p", "B_p", "C_p"])
phi = reshape_input(phi, expected)
```

*Phi is a PyTorch module that now fits into a normal training loop:*

```
...
semantic_loss = -(phi(y_pred).log()).mean()
loss = classification_loss + semantic_loss
```

### 3.2 GPU-Accelerated Circuits

A second key component of DeepLog is its use of GPU-accelerated circuits for evaluating Boolean formulas, probabilistic expressions, and related symbolic computations. These circuits provide an efficient inference structure for reasoning tasks that would otherwise be executed on the CPU, creating a performance bottleneck in hybrid systems. In DeepLog, we rely on the KLAY Maene et al. (2025) package for efficient circuit evaluation; comparative performance results are summarized in Table 1. To enforce structural properties such as determinism or decomposability, DeepLog targets specialized circuit backends, including representations based on sentential decision diagrams Darwiche (2011).

## 4 DeepLog as a Backend Architecture for NeSy Systems

Different NeSy systems appear fundamentally different, yet in practice they all execute the same pipeline: enumerate symbolic structures, attach neural scores, and aggregate them. This duplication obscures real design trade-offs and leads to

| Digits | Inference time [s/query] |                       |                       |
|--------|--------------------------|-----------------------|-----------------------|
|        | DPL CPU                  | DeepLog CPU           | DeepLog GPU           |
| 1      | $6.14 \times 10^{-3}$    | $1.09 \times 10^{-5}$ | $5.62 \times 10^{-7}$ |
| 2      | $3.52 \times 10^{-2}$    | $4.13 \times 10^{-5}$ | $1.52 \times 10^{-6}$ |
| 3      | $1.32 \times 10^{-1}$    | $2.43 \times 10^{-3}$ | $1.52 \times 10^{-4}$ |
| 4      | –                        | $1.96 \times 10^{-1}$ | $5.40 \times 10^{-2}$ |

Table 1: From Derkinderen et al. (2025), the average time in seconds to evaluate the neural network and arithmetic circuit for one query in the MNIST-Addition task. DeepProbLog (DPL) is included as a reference. DeepLog with GPU acceleration is most performant.

CPU-bound, hard-to-optimize implementations. At a semantic level, inference in virtually all NeSy systems reduces to evaluating a single formula De Smet and De Raedt (2025). Apparent differences arise from how this formula is calculated, not from what is computed. The DeepLog abstract machine Derkinderen et al. (2025) was introduced to make this computation explicit. It serves as a semantic intermediate representation and abstract machine target for NeSy systems, separating front-end symbolic languages from back-end execution strategies.

Languages such as DeepProbLog Manhaeve et al. (2018) and NeurASP Yang et al. (2023) use a Prolog- and ASP-like language, respectively. The base languages are lifted to the NeSy level through inclusion of the neural predicate, which parameterizes the probability of each model of the logic. Such a system would be implemented by first gathering proofs from a solver or engine. These proofs, whose symbolic information is backed by neural networks that provide their probabilities, are then used to calculate the probability of the query. While this process can be optimized (cf. knowledge compilation), without careful design the inference is typically still CPU-bound.

In DeepLog, the process is closer to Figure 1. The solver or engine does not construct a proof, but instead calls methods from a factory object in the DeepLog library, which immediately returns an instantiation of a DeepLog module. This factory can be configured to rely on certain components, or on techniques of choice (e.g. circuits versus sampling based approaches).

**Example 2.** *To demonstrate DeepLog’s support in developing neurosymbolic systems, we show how one could for example implement logic tensor network (LTN) Badreddine et al. (2022). LTN is a neuro-symbolic system where fuzzy predicate logic formulas are transformed into Tensorflow modules. In it, entities are represented with tensors, predicates are functions that map tensors onto fuzzy scores, and quantification is implemented using aggregation. Here we give the definitions of the separate components:*

```
ltm_fuzzy = {
    'not': lambda x: 1 - x,
    'and': lambda x, y: x * y,
    ...
}
exists = AggregationModule(name='exists',
    op=lambda x: p_mean(x, 6)
...
class EqualityPredicate(Predicate):
    functor, arity = 'eq', 2
```

```
structure = 'fuzzy'
```

```
def forward_predicate(self, x, y):
    return -(x - y).norm(dim=1).exp()
```

*These can now be combined into a single factory object, with a one-to-one mapping from LTN concepts to factory methods:*

```
ltm = DeepLogModuleFactory(
    structures = {'fuzzy': ltm_fuzzy},
    aggregators = {'exists': exists, ...},
    ...
    predicates = [EqualityPredicate],
)
Not = lambda x: ltm.unary_node('not', x)
Or = lambda x, y: ltm.binary_node('pr', x, y)
...
```

This configurability of the DeepLog backend allows us to easily digest the NeSy alphabet soup, as we can easily substitute probabilistic logic with fuzzy logic, and more generally compare inference techniques while keeping the remainder fixed.

## 5 Conclusion and future work

We introduced *DeepLog*, an operational framework that serves as a neurosymbolic abstract machine bridging symbolic reasoning and deep learning within standard PyTorch workflows. By separating semantic specification from operational realization, DeepLog enables automatic composition, validation, and efficient execution of neurosymbolic models on modern accelerators. Through symbolically annotated modules, a semantic intermediate language, and a compiler-style factory, the framework supports both ML practitioners seeking lightweight integration of symbolic constraints and NeSy developers building new languages and inference paradigms on a shared backend.

Several directions for future work remain. A primary focus is the automation of more aggressive circuit-level optimizations. While DeepLog already supports multiple compilation strategies and circuit backends, we aim to incorporate systematic optimization techniques Wang et al. (2024), including structure-aware rewritings, fusion of aggregation and inference, and others. Automating such transformations would further reduce manual engineering effort while improving performance and scalability. Beyond circuit optimization, future work includes providing alternative inference mechanisms such as sampling.

## Acknowledgements

This work project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (Grant agreement No. 101142702). This research received funding from the Flemish Government under the “Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen” programme.

## References

- K. Ahmed, T. Li, T. Ton, Q. Guo, K.-W. Chang, P. Kordjamshidi, V. Srikumar, G. Van den Broeck, and S. Singh. Pylon: A pytorch framework for learning with constraints. In *NeurIPS 2021 Competitions and Demonstrations Track*, pages 319–324. PMLR, 2022.
- S. Badreddine, A. d. Garcez, L. Serafini, and M. Spranger. Logic tensor networks. *Artificial Intelligence*, 303:103649, 2022.
- A. Darwiche. Sdd: A new canonical representation of propositional knowledge bases. In *IJCAI*, pages 819–826. IJCAI/AAAI, 2011.
- L. De Smet and L. De Raedt. Defining neurosymbolic ai. *arXiv preprint arXiv:2507.11127*, 2025.
- V. Derkinderen, R. Manhaeve, R. Adriaensen, L. Van Praet, L. De Smet, G. Marra, and L. De Raedt. The DeepLog Neurosymbolic Machine. *arXiv preprint arXiv:2508.13697*, 2025.
- A. d. Garcez and L. C. Lamb. Neurosymbolic ai: The 3rd wave. *Artificial Intelligence Review*, 56(11):12387–12406, 2023.
- P. Hitzler and M. K. Sarker. Neuro-symbolic artificial intelligence: The state of the art. 2022.
- J. Maene, V. Derkinderen, and P. Z. D. Martires. KLayer: Accelerating arithmetic circuits for neurosymbolic AI. In *ICLR*, 2025. URL <https://openreview.net/forum?id=Zes7Wyif8G>.
- R. Manhaeve, S. Dumančić, A. Kimmig, T. Demeester, and L. De Raedt. Deepproblog: Neural probabilistic logic programming. *Advances in neural information processing systems*, 31, 2018.
- G. Marra, F. Giannini, M. Diligenti, and M. Gori. Lyrics: A general interface layer to integrate logic inference and deep learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 283–298. Springer, 2019.
- G. Marra, S. Dumančić, R. Manhaeve, and L. De Raedt. From statistical relational to neurosymbolic artificial intelligence: A survey. *Artificial Intelligence*, 328:104062, 2024. ISSN 0004-3702. doi: <https://doi.org/10.1016/j.artint.2023.104062>. URL <https://www.sciencedirect.com/science/article/pii/S0004370223002084>.
- E. van Krieken, S. Badreddine, R. Manhaeve, and E. Giunchiglia. Uller: A unified language for learning and reasoning. In *International Conference on Neural-Symbolic Learning and Reasoning*, pages 219–239. Springer, 2024.
- B. Wang, D. Mauá, G. Van den Broeck, and Y. Choi. A compositional atlas for algebraic circuits. *Advances in Neural Information Processing Systems*, 37:141318–141355, 2024.
- J. Xu, Z. Zhang, T. Friedman, Y. Liang, and G. Broeck. A semantic loss function for deep learning with symbolic knowledge. In *International conference on machine learning*, pages 5502–5511. PMLR, 2018.
- Z. Yang, A. Ishay, and J. Lee. Neurasp: Embracing neural networks into answer set programming. *arXiv preprint arXiv:2307.07700*, 2023.