Neurosymbolic Artificial Intelligence
Volume 2: 1–19

DOI: 10.1177/29498732261443183
journals.sagepub.com/home/nai

Neuro-Symbolic Frameworks: Conceptual Characterization and Empirical Comparative Analysis

Sania Sinha[1,2], Tanawan Premsri[2], Danial Kamali[2] and Parisa Kordjamshidi[2]

Abstract

Neurosymbolic (NeSy) frameworks combine neural representations and learning with symbolic representations and reasoning. Combining the reasoning capacities, explainability, and interpretability of symbolic processing with the flexibility and power of neural computing allows us to solve complex problems with more reliability while being data-efficient. However, this recently growing topic poses a challenge to developers with its learning curve, lack of user-friendly tools, libraries, and unifying frameworks. In this paper, we characterize the technical facets of existing NeSy frameworks, such as the symbolic representation language, integration with neural models, and the underlying algorithms. A majority of the NeSy research focuses on algorithms instead of providing generic frameworks for declarative problem specification to leverage problem solving. To highlight the key aspects of NeSy modeling, we showcase three generic NeSy frameworks—*DeepProbLog*, *Scallop*, and *DomiKnowS*. We identify the challenges within each facet that lay the foundation for identifying the expressivity of each framework in solving a variety of problems. Building on this foundation, we aim to drive transformative action and encourage the community to rethink this problem in novel ways.

Keywords

Neurosymbolic, comparing neurosymbolic frameworks, DomiKnowS, DeepProbLog, Scallop, Combining learning and reasoning

Received: August 31, 2025; accepted: February 11, 2026

Editor: Leilani Gilpin, UC Santa Cruz, Department of Computer Science and Engineering; Eleonora Giunchiglia, Imperial College London, UK; Pascal Hitzler, Kansas State University, USA ;Emile van Krieken, University of Edinburgh, UK
Reviewers: Two anonymous reviewers

1 Introduction

Symbolic or *good old-fashioned* artificial intelligence (AI) focused on creating rule-based reasoning systems (Hayes-Roth, 1985) exemplified by early works such as the Physical Symbol System (Augusto, 2021; Newell, 1980) and ELIZA (Weizenbaum, 1966). However, drawbacks such as limited scalability due to the need to explicitly define domain predicates and rules for each task, lack of robustness in handling messy real-world data, and low computational efficiency led to a decline in the popularity of this paradigm, shifting the focus toward neural computing and deep learning. **Deep Learning** (Ahmad et al., 2019; LeCun et al., 2015) revolutionized AI, as nuanced relationships in data could be learned by backpropagation through multiple layers of processing and creating abstract representations of data. However, it led to a

[1]Mathematical Institute, University of Oxford, Oxford, UK
[2]Department of Computer Science and Engineering, Michigan State University, East Lansing, MI, USA

Corresponding Author:
Sania Sinha, University of Oxford, Mathematical Institute, Andrew Wiles Building, Woodstock Rd, Oxford, OX2 6GG, UK.
Email: sania.sinha@reuben.ox.ac.uk

loss of explainability (Li et al., 2023a), dependence on large amounts of data, lack of generalizability for unobserved situations, and rising concerns about its environmental sustainability (Bender et al., 2021). **Neurosymbolic (NeSy) AI** (Bhuyan et al., 2024; Hitzler & Sarker, 2021), a combination of symbolic AI and reasoning with neural networks, attempts to incorporate the capabilities of both worlds and create systems that are data and time efficient, generalizable, and explainable. NeSy models have been applied to several real-world applications (Bouneffouf & Aggarwal, 2022) in safety-critical areas (Lu et al., 2024) such as healthcare (Hossain & Chen, 2025) and autonomous driving (Sun et al., 2021). Several techniques have been proposed for this integration (Jayasingha et al., 2025; Kautz, 2022), trying to combine the advantages and mitigate the disadvantages of both symbolic and neural methods. However, due to the lack of unified libraries to facilitate this research and the focus on specific algorithms rather than generic frameworks, this research becomes less impactful. Moreover, the few generic frameworks tend to vary in problem formulation, implementation, algorithms, and flexibility of use. This poses a challenge in being able to compare their performance uniformly or identify a research direction that improves on previous work. To alleviate this issue, we provide a comparative study with the following key contributions, (a) Identifying the main components of existing NeSy frameworks, (b) Comparison of frameworks across the identified facets, (c) Highlighting the requirements for the next generation of NeSy frameworks, building upon the drawbacks of the current systems and the possible interplays between the neural and symbolic components. We plan to expand this study to cover more frameworks, while the three selected ones are used to explain the aspects of our characterization. These frameworks are demonstrated with four example tasks detailed in this paper tying the comparative facets concretely with a technical implementation.[1]

2 NeSy Frameworks

A NeSy framework should provide flexibility for modeling both neural and symbolic components and their interplay in a unified declarative framework, going beyond specific underlying algorithms and techniques (Kordjamshidi et al., 2016, 2015). On the symbolic side, a generic framework should support a symbolic representation language that can be seamlessly connected to neural components and cover various underlying symbolic reasoning mechanisms. On the neural side, we need to have the flexibility of connecting to various architectures, including various loss functions, sources of supervision, and training paradigms. More importantly, a NeSy framework should provide a modeling language for seamless integration of these two paradigms in building pipelines, joint decision-making models, and, in general, for arbitrary configurations of multiple models. Such a NeSy framework should support NeSy training and inference beyond specific integration algorithms. We distinguish between *NeSy techniques* and *NeSy frameworks*. By techniques, we mean when problem-specific algorithms are provided (Burattini et al., 2002; Lample & Charton, 2020). For example, **AlphaGo** (Silver et al., 2016) introduced a reinforcement learning solution to Go, using Monte Carlo Tree Search as a symbolic component inside a neural network. Another example is **NS-CL** (Mao et al., 2019) (NeSy Concept Learner) that integrates neural perception with symbolic reasoning to learn visual concepts and compositional language grounding for VQA tasks. Many other techniques and algorithms are proposed for the interplay between the two paradigms (Badreddine et al., 2022(@; Cohen et al., 2017; Lamb et al., 2020(@; Lima et al., 2005; Sathasivam, 2011; Serafini and d'Avila Garcez, 2016; Smolensky et al., 2016; Zhang et al., 2024) some of which are employed in the generic frameworks, such as Inference Masked Loss (Guo et al., 2020), Semantic Loss (Xu et al., 2018), Primal-Dual (Nandwani et al., 2019). NeSy techniques often lack the generality of frameworks, which are designed as broader tools intended for practical use and extensibility with new integration algorithms and with the capability of programming and configuring the two parts and their interplay.

In this work, we illustrate these components by focusing on a selection of representative, generic NeSy frameworks. These frameworks are chosen for their contribution to advancing the development of general-purpose approaches. **DeepProbLog** (Manhaeve et al., 2021) is a probabilistic logic programming language that incorporates neural predicates in logic programming with an underlying differentiable translation of logical reasoning. The probabilistic logic programming component is built on top of ProbLog (De Raedt et al., 2007). **DomiKnowS** (Faghihi et al., 2024, 2023; Rajaby Faghihi et al., 2021) is a declarative learning-based programming framework (Kordjamshidi et al., 2022) that integrates symbolic domain knowledge into deep learning. It is a Python framework that facilitates the incorporation of logical constraints that represent domain knowledge with neural learning in PyTorch. **Scallop** (Huang et al., 2021; Li et al., 2024, 2023b) is a framework that includes a flexible symbolic representation based on relational data modeling, using declarative logic programming similar to DeepProbLog while it is based on Datalog (Abiteboul et al., 1995) instead of Prolog. We also point to less general frameworks such as **LEFT** (Hsu et al., 2023) that is designed for grounding language in visual modality and compositional reasoning over concepts. The framework consists of a large language model (LLM) interpreter that converts natural language to logical programs. The generated programs are directed to a differentiable, domain-independent, and soft first-order logic-based executor. LEFT is limited to tasks requiring grounding language in

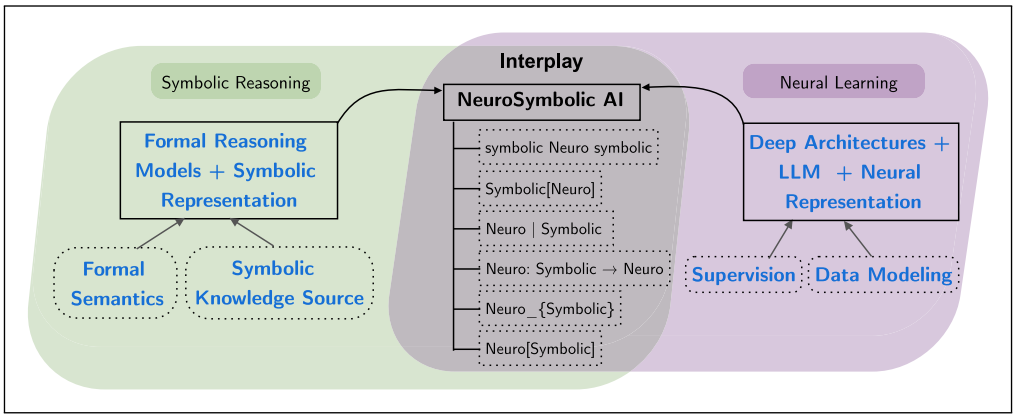

Figure 1. An overview of various components of a NeuroSymbolic artificial intelligence (AI) framework. NeuroSymbolic AI is at the intersection of symbolic reasoning and neural learning, depicted in green and purple, respectively. The interplay techniques can be classified into six types, shown in the middle. Arrows indicate the connections between components (not the computation flow).

vision such as visual question answering (Johnson et al., 2017; Liu et al., 2019; Yi et al., 2018). Building on this foundation, NeSyCoCo, (Kamali et al., 2025) has been introduced to address the limitations of LEFT, particularly the challenge of lexical variety of natural language and alleviating the issue of non-canonical predicate logic representations for handling unseen concepts. NeSyCoCo extends LEFT's approach by using distributed word representations to connect a wide variety of linguistically motivated predicates to neural modules, thus alleviating the reliance on a predefined predicate vocabulary. **PyReason** (Aditya et al., 2023) is a library built to support reasoning on top of outputs from neural networks. The neural component produces outputs such as labels or concept scores, while the symbolic component does graph-based reasoning with logical rules declared in a graph structure. It can produce an explanation trace for inference and has a memory-efficient implementation. **PLoT** (Wong et al., 2023) (Probabilistic Language of Thought) is a *proposed* framework leveraging neural and probabilistic modeling for generative world modeling. It models thinking with probabilistic programs and meaning construction with neural programs. The goal is to provide a language-driven unified thinking interface. **CCN+** (Giunchiglia et al., 2024) is a framework that modifies the output layer of a neural network to make results compliant with requirements that can be expressed in propositional logic. A requirement neural layer, ReqL, is built on top of the neural network. The standard cross-entropy loss is adapted into a ReqLoss to learn from the constraints in the ReqL layer. While CCN+ is a general framework, the underlying NeSy technique follows similar ideas that integrate logic in the architecture in (Li & Srikumar, 2019). **DeepLog** (Derkinderen et al., 2025) is another *proposed* NeSy AI framework that unifies logic and neural computation using a declarative paradigm. It introduces the DeepLog language, an annotated neural extension of grounded first-order logic that can abstract over various logics and apply them either within the model architecture or in the loss function. It employs computational algebraic circuits implemented on GPUs, forming a NeSy abstract machine. DeepLog allows efficient specification and execution of diverse NeSy models and inference tasks in a declarative fashion.

In this paper, we characterize the more generic NeSy frameworks based on: (a) Symbolic knowledge representation language, (b) Representation and flexibility of Neural Modeling, (c) Model Declaration, (d) Interplay between symbolic and sub-symbolic systems, and (e) The usage of LLMs. Figure 1 shows the connection between these aspects. The neural representations and the symbolic representations are the two main components of a NeSy framework. The neural representation guides learning and obtains supervision from the data, while the symbolic representations leverage symbolic reasoning and exploit symbolic knowledge during training or inference. Table 1 shows an overview of the frameworks across selected factors. In the next sections, we focus on **DomiKnowS**, **DeepProbLog**, and **Scallop** to provide a deeper investigation of the challenges in each component. Depending on the framework, certain tasks are easier to implement, while others require more effort. The chosen frameworks enable us to solve the same task in multiple frameworks.

3 Symbolic Knowledge Representation

Generic NeSy systems and frameworks use symbolic knowledge representation languages to encode constraints, facts, probabilities, and rules. Frameworks vary in how they represent and integrate this symbolic knowledge. Many employ classical formal logic, grounded in well-defined syntax and semantics, and adapt these representations and reasoning mechanisms within a unified integration framework. Some frameworks build on established formalisms such as logic

Table 1. Comparison of Frameworks and Key Factors.

	Symbolic			Interplay		
Framework	Lang	Knowledge Rep	Model Dec	Algorithm	Eff	LLM
CCN+	None	Propositional Logic	✗	Learning from constraints	✗	✗
DomiKnowS	None	Concepts & Relations graph, FOL	✓	Learning from constraints	✗	Faghihi et al. (2024)
DeepProbLog	ProbLog	FOL	✗	Deductive Reasoning	✗	✗
LEFT	None	FOL	✗	Differentiable soft logic	✗	Hsu et al. (2023)
PyReason	None	FOL	✗	Graph Reasoning	✓	✗
Scallop	DataLog	FOL	✗	Deductive Reasoning	✓	Li et al. (2024)

Lang: external language required; Knowledge Rep: knowledge representation; Model Dec: model declaration flexibility; Algorithm: supported algorithm(s) for learning and inference; Eff: computational efficiency considerations; LLM: use of large language models, FOL: first order logic.

Table 2. Comparison of Symbolic Representation Across Frameworks.

DomiKnowS

```
image = Concept(name='image')
digit = image(name='digits', ConceptClass=EnumConcept,
    values=digits)
image_pair = Concept(name='pair')
pair_d0, pair_d1 = image_pair.has_a(digit0=image, digit1=image)
s = image_pair(name='summations', ConceptClass=EnumConcept,
    values=summations)
# ...
sum_combinations.append(andL(getattr(digit, d0_nm)(path=('x',
    pair_d0)), getattr(digit, d1_nm)(path=('x', pair_d1))))
```

DeepProbLog

```
nn(mnist_net,[X],Y,[0,1,2,3,4,5,6,7,8,9]) :: digit(X,Y).
addition(X,Y,Z) :- digit(X,X2), digit(Y,Y2), Z is X2+Y2.
```

Scallop

```
self.scl_ctx = scallopy.ScallopContext(provenance=provenance,
    k=k)
self.scl_ctx.add_relation("digit_1", int,
    input_mapping=list(range(10)))
self.scl_ctx.add_relation("digit_2", int,
    input_mapping=list(range(10)))
self.scl_ctx.add_rule("sum_2(a + b) :- digit_1(a), digit_2(b)")
```

programming or constraint satisfaction. In contrast, others take an entirely new hybrid semantics, while preserving conventional symbolic syntax. Table 2 compares the implementation of symbolic knowledge (concepts or facts) for the MNIST Sum task. In general, the domain knowledge consists of the two concepts of *digits* and the *sum*.

As can be seen, DomiKnowS represents a part of symbolic domain knowledge as a graph $G(V, E)$, where the nodes are the concepts in the domain and the edges are the relationships between them. Each node can have properties. More complex knowledge beyond entities and relations is expressed with a pseudo first-order logical language with quantifiers designed in Python. DomiKnowS mostly interprets the symbolic knowledge as logical constraints, such as the implementation of `sum_combinations` in the given example. Unlike the other frameworks, DomiKnowS does not build on predefined formal semantics. It follows a FOL-like syntax for symbolic logical representations, making it independent of the formal semantics of an underlying formal language and allows more flexibility of representations and adaptation to underlying algorithms in the framework. DeepProbLog, on the other hand, utilizes logical predicates that are originally a part of the

probabilistic logic programs (Ng & Subrahmanian, 1992) of ProbLog (De Raedt et al., 2007), for its symbolic representation. These neural predicates obtain their probability distributions from the underlying neural models. Probabilistic facts, neural facts, and neural annotated disjunctions (nADs) whose probabilities are supplied by the neural component of the program can be added. Here, `digit` is a neural predicate as indicated by the use of `nn(⋯ )`. DeepProbLog follows the formal semantics of Prolog (Clocksin & Mellish, 2003), followed by ProbLog, its probabilistic extension. Finally, Scallop adopts a relational data model for symbolic knowledge representation (Kolaitis & Vardi, 1990). Scallop is built on top of the syntax and formal semantics of Datalog and its probabilistic extensions, relaxing the exact semantics of ProbLog. It allows for the expression of common reasoning, such as aggregation, negation, and recursion. Similar to DeepProbLog, some of these predicates in the symbolic part obtain their probability distribution from neural models, such as `digit_1` and `digit_2`. Additionally, while ProbLog requires exhaustive search for computations, Datalog can use top-k results and exploit database optimizations, making Scallop algorithmically more time-efficient than DeepProbLog.

4 Neural Models Representations

The other core component of a NeSy system is the neural modeling that is integrated with the symbolic knowledge discussed above. The neural models are mostly wrapped up under the logical predicate names in most of the frameworks that have an explicit logical knowledge representation language. To best leverage the reasoning capabilities of the symbolic system available and the ability of neural models to learn abstract representations from data, the neural models are used as abstract concept learners for the concepts defined as logical predicates in the symbolic representation. The neural model representation is often used to predict probability distributions for the symbolic concepts based on raw sensory inputs. Given that NeSy frameworks are explicit about the symbolic concepts and relations, reading data into the symbolic representation requires special considerations for processing raw data, putting data into structure, and **Data Modeling**. The aspect of data modeling is often ignored in current Nesy frameworks and it is done by the programmers in an ad hoc pre-processing step. The neural modeling is often written using standard deep learning libraries, such as PyTorch (Paszke et al., 2019).

Table 3 shows snippets of neural modeling expressions across frameworks, highlighting differences in implementation. Scallop utilizes relatively standard neural modeling using PyTorch, while needing an added context of symbolic rules. Although integrated into Python, the context relation and rule setup are verbatim from DataLog and only passed as a parameter to a function, which requires familiarity with DataLog and its semantics. DeepProbLog, on the other hand, needs manual configuration of the raw data and processing into queries built for ProbLog, on top of other standard neural components. This processed data is passed into the neural network which is then connected to a ProbLog program, such as `addition.pl` in the figure. DomiKnowS's neural component is built in PyTorch. Unlike other frameworks, DomiKnowS has built-in components called *Readers, Sensors* and *Module learners* for a more *transparent data modeling*, making the connection to neural components and feeding data into them explicitly in the program. This provides more flexibility in connecting the concepts to deterministic or probabilistic functions that can interact with other symbolic concepts. The module learner can also use custom models. This makes the interaction with raw data structured, transparent, and controllable.

5 Model Declaration

Most frameworks utilize neural components as abstract concept learners and use a symbolic component to reason over the learned concepts. Each learner is a model and *model declaration* refers to the flexibility of modularizing and connecting different learners. Each learner can receive supervision independently. In most NeSy frameworks, the supervision from data is usually provided based on the final output of the end-to-end model. For example, in an MNIST Sum task used throughout and detailed in Section 8, the neural and symbolic components are trained based on the final output of the sum, without access to individual digit labels in a semi-supervised setting. The task loss, for example, a Cross-Entropy Loss, is computed, and errors are backpropagated through the differentiable operations that led to the output generation. For example, in DeepProbLog, we can declare a single loss function associated with the entire neural component. Gradient computations differ across frameworks depending on whether losses are defined individually for each neural output or specified as a single global loss function. However, there remains a need for models capable of incorporating supervision at multiple levels of their symbolic representations. In DomiKnowS, loss computation can be defined for each symbol. Since each concept is linked to both learning modules and ground-truth labels, their losses can be integrated seamlessly. This enables joint training of all concepts alongside the target task, allowing each concept to be optimized more effectively—leveraging available data without relying solely on the target task's output. In other words, it provides the flexibility of building pipelines of decision making, obtaining distant supervision in addition to joint training and inference. Model

Table 3. For Neural Integration, DomiKnowS Utilizes Sensors and Readers for Reading in Data, while a Learner Connects to a Network.

DomiKnowS

```
class Net():
    # Neural Network
image_batch['pixels', image_contains.reversed] =
    JointSensor(image['pixels'], forward=make_batch)
image['logits'] = ModuleLearner('pixels', module=Net())
# ...
```

DeepProbLog

```
class MNIST_Net():
    # Neural Network
network = MNIST_Net()
net = Network(network, "mnist_net", batching=True)
net.optimizer = torch.optim.Adam(network.parameters(), lr=1e-3)
model = Model("models/addition.pl", [net])
# ...
```

Scallop

```
class MNISTSum2Net(nn.Module):
    def __init__(self, provenance, k):
        # Neural Network
        self.mnist_net = MNISTNet()
        # Scallop Context
        self.scl_ctx =
    scallopy.ScallopContext(provenance=provenance, k=k)
    # ...
class Trainer():
    def __init__(self, train_loader, test_loader, model_dir,
    learning_rate, loss, k, provenance):
        self.model_dir = model_dir
        self.network = MNISTSum2Net(provenance, k)
        self.optimizer = optim.Adam(self.network.parameters(),
    lr=learning_rate)
    # ...
```

DeepProbLog connects the neural network to the ProbLog file, requiring data handling to construct the terms and queries from the raw data. Scallop has an additional layer on top of the standard network that adds the symbolic context.

declaration is very well manifested in modern tools and libraries for building agentic frameworks, where multiple agents are structured into pipelines or graphs to make communicative decisions by passing the output of one agent as input to another (Plaat et al., 2025).

6 Interplay Between Symbolic and Subsymbolic

Kautz (2022) provides a characterization of the possible interplays between symbolic and sub-symbolic components. This interplay of NeSy can be explained by the concept of System 1 and System 2 thinking described in Kahneman (2011). Research in this field aims to create an ideal integration that seamlessly supports "thinking fast and slow" (Booch et al., 2021; Fabiano et al., 2023). Here, System 1 refers to the fast neural processing, while System 2 corresponds to the slower, more deliberate symbolic reasoning. Different methods for the integration of symbolic reasoning and neural programming have been explored such as employing *logical constraint satisfaction, integer linear programming (ILP), differentiable reasoning, and probabilistic logic programming.* In this section, we will discuss a system-level algorithmic comparison of the different frameworks.

Table 4. Time and Space Efficiency for Each Framework Across 4 Tasks.

	MNIST Sum			Toy-NER		
Framework	Training Time (ms)	Testing Time (ms)	Memory (MB)	Training Time (ms)	Testing Time (ms)	Memory (MB)
DomiKnowS	37.72	**2.34**	**573.80**	14.86	14.29	576.82
DeepProbLog	**5.84**	3.24	739.72	20.74	20.47	3767.08
Scallop	6.50	**2.35**	**581.36**	**1.50**	**1.05**	**297.1**
	Math-Inference			Simple VQA		
Framework	Training Time (ms)	Testing Time (ms)	Memory (MB)	Training Time (ms)	Testing Time (ms)	Memory (MB)
DomiKnowS	77.46	69.58	1039.30	81.02	60.52	1413.44
DeepProbLog	5.54	5.07	1588.30	755.13	363.81	1346.23
Scallop	**0.948**	**0.223**	**345.48**	**13.17**	**22.66**	**768.4**
LEFT	N/A	N/A	N/A	6.19	3.44	755.1

Time and memory records are averaged over 5 runs. Times are in milliseconds per sample. Memory utilization is in megabytes and indicates the overall memory for training.

DomiKnowS models the inference as an ILP problem to enforce the model to follow constraints expressed in first-order logical form (Van Hentenryck et al., 1992). The objective of the program is guided by the neural components, and the framework supports multiple training algorithms for learning from constraints. The Primal-Dual formulation (Nandwani et al., 2019) converts the constrained optimization problem into a min-max optimization with Lagrangian multipliers for each constraint, augmenting the original loss with a soft logic surrogate to minimize constraint violations. Sampling-Loss (Ahmed et al., 2022), inspired by semantic loss (Xu et al., 2018) and samples a set of assignments for each variable based on the probability distribution of the neural modules' output. ILP(Cropper & Dumančić, 2022) formulates an optimization objective based on Inference-Masked Loss(Guo et al., 2020) to constrain the model during training. The training goal is to adjust the neural models to produce legitimate outputs that adhere to the given constraints. At prediction time, ILP can also be applied to enforce final predictions that comply with given constraints. DomiKnowS relies on the off-the-shelf optimization solver Gurobi (Gurobi Optimization, LLC, 2024). **DeepProbLog** models each problem as a probabilistic logical program that consists of neural facts, probabilistic facts, neural predicates, and a set of logical rules. A joint optimization of the parameters of the logic program is done alongside the parameters of the neural component. Neural network training is done using learning from entailment (Frazier & Pitt, 1993) while in ProbLog, gradient-based optimization is performed on the underlying generated Arithmetic Circuits (Shpilka & Yehudayoff, 2010), which is a differentiable structure. The Arithmetic Circuits are transformed from a Sentential Decision Diagram (Darwiche, 2011) generated by ProbLog. Algebraic ProbLog (Kimmig et al., 2011) is used to compute the gradient alongside probabilities using semirings (Eisner, 2002). **Scallop** is similar in its setup to DeepProbLog, where it creates an end-to-end differentiable framework combining a symbolic reasoning component with a neural modeling component. They aim to relax the formal semantics required by the use of ProbLog in DeepProbLog and instead rely on a symbolic reasoning language extending DataLog, built into their framework. They have a customizable provenance semiring framework (Green et al., 2007), where different provenance semirings, such as the extended max-min semiring and the top-k proofs semiring, allow learning using different types of heuristics for gradient calculations. Table 4 compares the computational efficiency of these models at training and inference time on a single training/testing example. As theoretically suggested, Scallop is expected to outperform other frameworks in inference and training speed, owing to its memory and time-efficient implementation in Rust. The results in Table 4 support this expectation, with Scallop achieving the fastest inference time, on par with DomiKnowS. In practice, DeepProbLog achieves slightly faster training performance than Scallop. This discrepancy may be due to overhead unrelated to the core algorithmic complexity. DomiKnowS exhibits slower training due to the overhead of uploading the entire graph of data into memory.

7 Role of LLMs

Large foundation models hold significant promise for overcoming the bottleneck of acquiring symbolic representations, which are essential for symbolic reasoning and consequently in NeSy frameworks.

Source of Symbolic Knowledge: The symbolic knowledge in NeSy systems, which is integrated with the neural component, can originate from several distinct sources. While most systems require explicit, hand-crafted symbolic knowledge, earlier classical logic-based learning research can be used for automatically learning rules from data by using inductive

logic programming (Bratko & Muggleton, 1995; Nienhuys-Cheng & de Wolf, 1997) or mining constraints. Nowadays, even LLMs can be utilized to generate symbolic knowledge (Acharya et al., 2024; Mirzaee & Kordjamshidi, 2023; Pan et al., 2023a; Xu et al., 2024a). Several NeSy frameworks and systems have tried utilizing large foundation models to generate the symbolic knowledge, based on the task or query, to overcome the labor-intensive nature of hand-crafting rules for every single task and the time required in the automatic learning of symbolic knowledge from data (Ishay et al., 2023; Xu et al., 2024a; Yang et al., 2024). Extraction of symbolic representations from Foundation Models has become possible given the vast implicit knowledge stored within these models, such as LLMs and multimodal models, which are trained on massive and diverse corpora (Li et al., 2024; Petroni et al., 2019). These models can generate symbolic content (e.g., candidate rules, knowledge graph triples, or logic statements), perform reasoning that mimics symbolic inference, or act as components alongside symbolic modules (Fang & Yu, 2024). For example, LLMs can be prompted to extract facts from unstructured text, effectively populating a symbolic knowledge graph (Yao et al., 2025). Techniques like Symbolic Chain-of-Thought inject formal logic into the LLM's reasoning process, improving accuracy and explainability on logical reasoning tasks (Xu et al., 2024b). However, foundation models are prone to hallucinations and lack the strict logical guarantees of traditional symbolic systems (Zheng et al., 2024). Therefore, integrating foundation models often requires careful prompting, verification steps to ensure reliability (Xu et al., 2024b). **Generation of inputs to symbolic engines:** LLMs have also been used to generate translations from raw inputs, specially natural language, to symbolic language that is then fed into a symbolic reasoner. In examples such as Logic-LM (Pan et al., 2023b), LLMs are leveraged to convert a natural language query into symbolic language that is then solved by a symbolic reasoner. This method improves the performance of unfinetuned LLMs on logical reasoning-based tasks. DomiKnowS (Faghihi et al., 2024) takes this a step further by enabling users to describe problems in natural language, which LLMs then use to generate relevant concepts and relationships. Through a user-interactive process, these concepts and relationships are refined iteratively. Finally, the LLM translates the user-defined constraints from natural language into first-order logic representations before converting them into DomiKnowS syntax. Some systems use LLMs in multiple capacities. In VIERA (Li et al., 2024), which is built on top of Scallop, 12 foundation models can be used as plugins. These models are treated as stateless functions with relational inputs and outputs. These foundation models can be either language models like GPT (OpenA et al., 2024) and LLaMA (Touvron et al., 2023), vision models such as OWL-ViT (Minderer et al., 2022) and SAM (Kirillov et al., 2023), or multimodal models such as CLIP (Radford et al., 2021). These models can be used to extract facts, assign probabilities, or for classification, and are treated as "foreign predicates" in their interface. An older version, DSR-LM (Zhang et al., 2023) of this utilized BERT-based language models for perception and relation extraction, combined with a symbolic reasoner for question answering. LEFT, on the other hand, uses LLMs both for the generation of the concepts that are used for grounding and as an interpreter to generate the first-order logic program corresponding to a natural language query, that is solved by the symbolic executor.

8 Example Tasks

NeSy frameworks formulate problems in various ways based on their implementation and symbolic interpretation. In **DomiKnowS**, the symbolic reasoning part is formulated as a logical constraint solving problem. The domain is represented as a graph $G(V, E)$, where the nodes are the concepts in the domain and the edges are the relationships between them. Each node can have properties. The final logical constraints apply to the graph concepts. In **DeepProbLog**, the symbolic reasoning problem is interpreted as probabilistic logic programs in ProbLog. In **Scallop**, similarly, the problem is viewed as a combination of the neural and the symbolic components, where the symbolic part is a probabilistic logical program similar to DeepProblog with further optimized inference. In **LEFT**, the problem is limited to the application of concept learning and grounding language into the visual modality. Here, the neural model is composed of feature extractors, object and relation classifiers (concept learners), and a first-order logic program generator for a given question. In this section, we will compare the problem formulations in each of these frameworks for a set of tasks. **Note** that we only include LEFT for the visual question answering task due to the domain-specific nature of the framework. All code associated with these tasks and referenced in this section is maintained publicly on GitHub.[1]

8.1 MNIST Sum

The MNIST Sum task is an extension of the classic MNIST handwritten digit recognition task (Lecun et al., 1998), where given two images of digits, the task is to output their sum that is a whole number. The training examples consist of the two images of the digits and the ground-truth label of their sum. The individual labels of the digits are not available for training.

8.1.1 DomiKnowS.

Problem Specification. DomiKnowS formulates the problem using graph representations of concepts, relations, and logic. For performing the MNIST Sum task in DomiKnowS, the first concept defined is *image* concept representing visual information. The *digit* concept, a subclass of image, is introduced to represent the output class, ranging from 0 to 9. To establish relationships between digit images, the *image pair* concept is defined as an edge connecting two digit concepts. The sum concept is then introduced under the image pair to represent the summation of the two digit concepts and the ground-truth output of the program. For this task, three constraints are defined. The first two constraints utilize *exactL* to ensure that the predicted digit and sum values belong to only one valid class. Another constraint enforces that the expected sum value matches the sum of the two digit predictions. This is implemented using *ifL* constraints, which verify whether the predicted digits form one of the possible solutions for a valid sum. If multiple solutions exist, the *orL* constraint ensures that at least one of the answers corresponds to the predicted digits.

Neural Modeling. The model declaration comprises standard neural modeling components, including data loading, pre-processing, neural network definition, and loss function specification. The process begins with the *ReaderSensor*, which reads the input image. Next, a relation concept is defined using another sensor, *JointSensor*, to establish connections between images. The module learner is then employed to generate an initial prediction for the digit concept, which is subsequently passed to another sensor, *FunctionalSensor*, to compute the sum of two images.

8.1.2 DeepProbLog.

Problem Specification. DeepProbLog formulates a problem regarding probabilistic facts, neural facts, and nADs. In the MNIST Sum task, the fact X is defined to represent the input image. A neural network function is then introduced to map X to its corresponding digit, denoted as digit(X, Y). To enforce constraints about the summation and the ground-truth sum, a function is defined to compute the sum of two digits.

Neural Modeling. The neural modeling follows a standard neural network setup, such as a CNN-based classifier. It is preceded by data loading and pre-processing, which are performed separately from the ProbLog program. Thus, the neural model used in DeepProbLog can be initialized independently of the DeepProbLog model. Once the neural model is initialized, the framework passes it along with a probabilistic program as input.

8.1.3 Scallop.

Problem Specification. Scallop formulates the problem in terms of relations, values, and (Horn) rules derived from Datalog. As discussed earlier, the concepts and constraints defined in this framework are similar to those in DeepProbLog. However, these rules can be directly embedded into a Scallop program through its API. The process begins by establishing the concepts *digit1* and *digit2* to represent the digit values of two given images. Based on the summation of these two values, it must be equal to the *sum*_2 logical reasoning module, which serves as the ground truth for this task.

Neural Modeling. Unlike DeepProbLog, the neural modeling is integrated with Scallop's relation and rule declaration. The neural modeling remains a standard neural network.

8.2 Shapes

The **Shapes** dataset is a synthetic VQA benchmark designed to evaluate elementary spatial reasoning. Each sample consists of a 128×128 pixel image where the task is to answer the fixed question: "Is there a red shape above a blue circle?". The primary rationale for using a synthetic dataset is to create a controlled experimental environment. This approach allows us to isolate specific reasoning skills—such as attribute binding and relational understanding—from the perceptual complexities and spurious correlations.

Positive examples are generated by programmatically placing a red object (circle, square, or triangle) above a blue circle. In negative examples, this specific spatial configuration is absent. To increase task complexity, every image also contains one to three randomly placed, non-overlapping distractor objects with varying shapes and colors. The dataset comprises 2,000 images, divided into perfectly balanced training and testing sets of 1,000 samples each. Examples of this benchmark are shown in Figure 2.

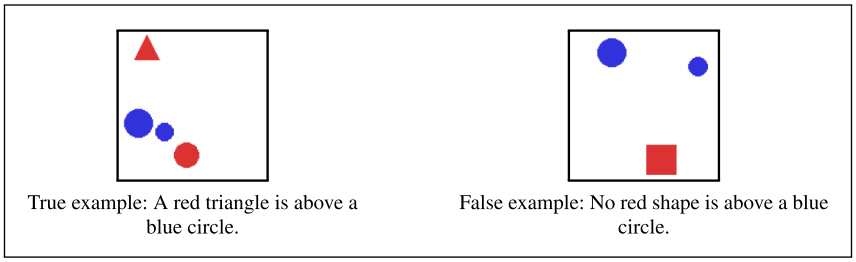

Figure 2. Examples from the shapes dataset for the question "Is there a red shape above a blue circle?" True example: A red triangle is above a blue circle and False example: No red shape is above a blue circle.

8.2.1 Common Perception—Reasoning Interface. We decompose the system into: (i) a neural perception unit producing distributions over object attributes and pairwise relations; and (ii) a symbolic reasoning unit that consumes these as soft facts over the object domain and evaluates a single existential rule encoding the query. Supervision is a binary yes/no label optimized via a cross-entropy loss on the final answer distribution.

8.2.2 DomiKnowS.

Problem Specification. DomiKnowS formulates the problem as a graph representation. It first declares the *image* concept to represent the visual input. The *objects* concept is then defined to represent the individual objects contained within the image. An explicit connection declaration is established in the framework between images and objects, indicating that each image may contain multiple objects. Under the image concept, two additional concepts—*color* and *shape*—are introduced to represent the attributes of each object for each shape and color considered. These two concepts serve as the outputs of the neural modeling component. Next, DomiKnowS defines a *relation* concept between pairs of objects to capture their relational structure within the same image. Finally, symbolic reasoning is expressed using *existsL*, which denotes the existence of a particular combination of queries. The inference takes the following form:

```
existsL(is_red(X), is_blue(Y), is_circle(Y), relation(Rel))
```

where the X and Y represent the first object and the second object, *Rel* represent the relation of X and Y.

Neural Modeling. DomiKnowS uses a class of functions called *ReaderSensor*s to process the raw input data. A *ModuleLearner* is then employed to query the object-centric encoder, producing representations of the objects within the image. These representations are subsequently passed to three *ModuleLearner* to predict the color (red and blue) and shape (circle) attributes of the objects. A *CompositionCandidateSensor* is used to connect pairs of objects within the image, providing the information needed to compute their relations. Finally, the framework aggregates all information through predefined logical expressions, which are used to infer the final output.

8.2.3 DeepProbLog.

Problem Specification. We use a ProbLog program with nADs for `color(Image,O,C)`, `shape(Image,O,S)`, and `relation(Image,O1,O2,R)`. The decision rule is:

```
answer(Image, yes) :- obj(O1), obj(O2),
 O1 /= O2,
 color(Image,O1,red),
 color(Image,O2,blue),
 shape(Image,O2,circle),
 relation(Image,O1,O2,R).
```

A complementary `answer(Image, no)` ensures a normalized binary outcome. The object domain `obj/1` is declared up to $n_{\max}$ derived from annotations. Batched queries are issued as `answer(tensor(batch(I)), Y)` and solved with an exact engine.

Neural Modeling. An object-centric encoder crops and embeds objects; heads output categorical distributions for color and shape per object and for relation per ordered object pair. These heads are wrapped as DeepProbLog networks and bound to the program's neural predicates, enabling end-to-end training from the answer predicate.

8.2.4 Scallop.

Problem Specification. We declare unary predicates over object indices for attributes (`red(i)`, `blue(j)`, `circle(j)`) and a binary predicate for the chosen spatial relation `R(i,j)`. The query is encoded as a Horn rule:

```
ans():−red(i),blue(j),circle(j),above(i, j).
```

Scallop maps soft facts to weighted relations under a differentiable provenance and produces a normalized {`yes`,`no`} answer.

Neural Modeling. Per-object (color, shape) and per-pair (relation) distributions from the encoder are converted to Scallop facts, respecting masks for padded objects and pairs. The Scallop context composes these facts with the rule to yield the answer distribution, providing gradients to the neural unit.

8.2.5 LEFT.

Problem Specification. We express the query in first-order logic executed by LEFT's generalized FOL executor:

```
exists(Object, lambda x: exists(Object, lambda y:
above(y, x) and red(x) and circle(y) and blue(y)))
```

The executor grounds over the object domain and evaluates the formula using attribute predicates for unary properties and a binary predicate for the relation, returning a scalar decision consistent with the query.

Neural Modeling. The object-centric encoder yields per-object attribute scores (for color/shape) and per-pair relation scores. These tensors parameterize the executor's predicates, forming a differentiable perception–reasoning pipeline trained on the binary supervision.

8.3 Toy NER

For a simplified, toy version of the Named-Entity Recognition task (Tjong Kim Sang and De Meulder, 2003), we create a dataset of randomly generated embeddings representing persons and locations. The objective is to learn the concepts of "works_in," that is, whether a person works in a location, and "is_real_person," that is, whether a given embedding is a person or not. These concepts are learned using indirect supervision on two queries, that compose the atomic concepts.

```
constraint1 = is_real_person(P1) AND works_in(P1,L1) AND is_real_person(P2)
  AND works_in(P2,L2)
constraint2 = is_real_person(P2) AND works_in(P2,L2)
  OR is_real_person(P3) AND works_in(P3,L3)
```

Here, $P1, P2, P3, L1, L2, L3$ are input embeddings with Ps representing persons and Ls representing locations. Thus, the final task is to predict the output of these two constraints (true or false), given 6 embeddings corresponding to 3 persons and 3 locations.

8.3.1 DomiKnowS.

Problem Specification. To perform the Toy NER task, DomiKnowS begins by constructing a graph representation of the problem. It first defines the *person* and *location* concepts to represent the entity types of interest. Under the *person* concept, three sub-concepts are declared to represent three individual persons, and under the location concept, three sub-concepts are introduced to represent distinct locations. A *pair* relation is then defined to connect a person and a location. This relation is used to create three separate *work_in[i]* concepts, each representing the output relation between person *i* and location *i*. Finally, the inference queries are expressed in a form-based manner, using *andL* to represent logical conjunctions (AND) between concepts and *orL* to represent logical disjunctions (OR) in the desired queries above.

Neural Modeling. The model declaration follows the standard pipeline of neural components. It begins with a *ReaderSensor* to process the raw input data. A *ModuleLearner* is then invoked, using the embeddings of either person or location to generate predictions based on the given representations. A *JointSensor* is subsequently employed to connect the person and location concepts in order to form the *work_in[i]* relation specified in the problem definition. Finally, model inference is carried out based on predefined queries, yielding the final output.

8.3.2 DeepProbLog.

Problem Specification. DeepProbLog declares 6 concepts, *is_real_person[i]* for each person i and *works_in[i]* for each location *i*. These concepts are used to formulate the two constraints with the final query, *check*, being the conjunction of the two. Note that DeepProbLog does not support multiple simultaneous queries and hence, the two constraints cannot be trained with individual supervision for labels of each.

Neural Modeling. DeepProbLog uses two neural networks for the classification of the two concepts. The main challenge is the creation of the dataloader that reads from the raw data and converts it into the format necessary to send queries with the appropriate value substitutions for variables. This integration with ProbLog needs to be done by the user from scratch.

8.3.3 Scallop.

Problem Specification. Scallop utilizes 6 concepts similar to DeepProbLog. The value of the two constraints is merged using a final module *check(P1* P2 * W1 * W2 + P3 * W3)*. Here, the results of concepts *is_real_person[i]* (P1, P2, P3) and *works_in[i]* (W1, W2, W3) are passed instead of the embeddings themselves. This context is passed through Python itself, instead of a separate Datalog file.

Neural Modeling. The neural modeling component is standard with flexible loss declarations. To adapt from the raw data, an embedding generator *NERDataset* is created which is the same as the *JSONDataset* utilized in the DeepProbLog solution. However, there is no need to manually adapt the data to generate queries to the datalog component, as the forward function of the neural network is the final query and only requires the outputs of the neural networks passed to it.

8.4 Math Equation Inference

The Math Equation Inference task is designed to evaluate whether a neural network can learn local mathematical concepts from global supervision. The input to this task consists of two lists, each containing six real numbers sampled uniformly from the range $[-1, 1]$. The objective is to determine whether the model can learn from a global condition that encompasses the properties of the first list, the properties of the second list, and the relationship between them. We consider two properties, that is, $\sum_{i=0}^{8} x_i > 0$ and $\sum_{i=0}^{8} |x_i| > 0.5$. For relations between the two lists, we examine two cases: (i) whether the first elements of the lists have the same sign, and (ii) whether their last elements have opposite signs. These concepts are learned through indirect supervision using global queries that compose the atomic concepts, expressed as follows:

```
property1(L1) AND property2(L2) AND relation(L1, L2)
```

where $L1, L2$ are randomly generated lists of six real numbers, property1 and property2 are drawn from the set of considered properties, and relation is drawn from the set of considered relations.

8.4.1 DomiKnowS.

Problem Specification. For implementing this task in DomiKnowS, the first concept defined is the *problem* concept, which represents the overall mathematical inference problem. Next, the *lst* concept is introduced to represent a list of considered numbers. Under this *lst* concept, all possible condition concepts–*is_cond1* and *is_cond2*–are defined to represent the output class of each list. Subsequently, two relation concepts, *is_relation1* and *is_relation2*, are introduced between pairs of *lst* instances to capture the two relational conditions described in the problem statement. Importantly, all possible conditions and relations must be defined, even if some are not ultimately used in the final inference. To define the inference query, the logical operator andL is employed to combine the three relevant concepts. For example: *andL(is_cond1(L1), is_relation1(L1, L2), is_cond2(L2))*. This query corresponds to the case where the sum of $L1$ is greater than 0 (*is_cond1*), the first elements of $L1$ and $L2$ share the same sign (*is_relation1*), and the sum of the absolute values of $L2$ is greater than 0.5 (*is_cond2*).

Neural Modeling. DomiKnowS begins with a *ReaderSensor* to process the two input lists of numbers. Then, *CompositionCandidateSensor* is employed to connect the two lists, forming the intermediate representation required for relation prediction in later stages. Four neural networks are instantiated using *ModuleLearner*—two dedicated to property concepts and two to relation concepts. These networks are trained to predict the respective properties and relations. The predicted concepts are subsequently integrated into the final query, which specifies the target relation for the given problem and serves as the prediction label during training and evaluation. It is important to note that DomiKnowS requires all possible inference queries to be explicitly represented in the graph. Each query is set as either active or inactive to obtain the correct prediction during inference.

8.4.2 DeepProbLog.

Problem Specification. In this task, DeepProbLog defines four neural predicates: Two for the considered properties (*is_property[i]_nn* for object property i), and two for the considered relations (*is_relation[i]_nn* for relation i). These four neural predicates are then combined to define the final query concept, *inference*. One *inference* concept is created for each possible combination of the condition of $L1$ (2 possibilities), the condition of $L2$ (2 possibilities), and the relation between $L1$ and $L2$ (2 possibilities). In total, eight inference concepts are formulated. During evaluation, only the relevant concept corresponding to the input configuration is activated and used to produce the final prediction for the problem.

Neural Modeling. DeepProbLog employs four neural networks, corresponding to the defined concepts: Two for property concepts and two for relation concepts. The pipeline then proceeds with data loading, where all possible candidate lists of numbers are read separately and pre-processed to obtain the required model inputs. This also includes manually connecting the list of numbers within the same problem. Then, all defined neural networks are called to obtain all possible relations and properties output. Lastly, the query must be constructed in the exact predefined pattern to ensure that the system produces the correct output aligned with the desired inference condition.

8.4.3 Scallop.

Problem Specification. Similar to DomiKnowS and DeepProbLog on this task, Scallop begins with the declaration of four concepts, that is, two property concepts and two relation concepts. However, Scallop only defines the one final module, *inference(P1 * P2 * R)*, that accumulates the probability based on the output of the properties of $L1$ and $L2$, and the relation between $L1$ and $L2$. The output is defined in the neural modeling part to get the connection between the final module and the defined concept.

Neural Modeling. The neural modeling component in Scallop follows the standard pipeline. However, Scallop requires a manually defined connection between the model output and the declared final module. This requirement arises because the final module is specified in a general form, rather than being tied to a particular combination of inference concepts. In contrast, other frameworks explicitly define every possible query corresponding to a combination of properties and relations and refer only to those queries during inference.

9 Discussion and Future Direction

Table 1 summarizes the comparative aspects of existing frameworks and outlines future directions for optimizing, as we observe many columns marked with "✗," implying most frameworks present challenges that hinder the application and

flexibility of the frameworks. While current frameworks are functional, future developments should take a more holistic approach that considers all aspects from an end-user perspective, aiming to improve usability as general-purpose libraries and foster wider adoption of NeSy methods.

Symbolic Representation The generic NeSy frameworks provide a formal knowledge representation language of their choice. The selected languages are often based on pure logical formalisms with established formal semantics, for example, Datalog or Prolog. However, we argue that knowledge representation for NeSy frameworks needs to be an innovative language designed for this integration purpose with adaptable semantics with learning as the pivotal concept (Kordjamshidi et al., 2022). Restricting these frameworks to classical AI formalisms and formal semantics limits the level of extension that can be made and restricts the support of various algorithms and types of integration.

Neural Modeling Most of the examined frameworks leave neural modeling and the task of connecting the symbolic and sub-symbolic components, up to the user. This connection usually requires low-level data pre-processing, which is time consuming to implement. A lack of user-friendly libraries discourages developers from using NeSy methods to solve downstream tasks. There is a need for abstractions in these frameworks (Kordjamshidi et al., 2022) that improve user experience and remove the need for users to implement such data processing from scratch.

Model Declaration. There is a need to be explicit about the low-level components of the neural architecture, enabling us to design interactions between neural and symbolic components and connect them as intended. The goal is to provide flexibility in designing arbitrary loss functions and connecting them to data for supervising concepts at various neural layers, which will allow any symbol to be learnable.

Types of Interplay. Considering Kautz (2022)'s classification, current frameworks are limited in supporting one or two ways of interactions. DeepProbLog and Scallop utilize one form of implementation, while DomiKnowS has multiple settings as detailed in Section 6. One of the key challenges is determining the appropriate level of abstraction in a neural model after which reasoning should occur. The classification types demonstrate how a neural model can identify the relevant symbolic representations and suggest that NeSy frameworks could leverage these models to learn and route inputs to the corresponding symbolic reasoning system. However, it remains unclear what level of abstraction is most effective for solving the end task in practice.

LLM. Drawbacks often associated with employing symbolic AI into neural computing, such as the creation of symbolic knowledge for integration, can be mediated with the use of LLMs and foundation models. LLMs have the potential to alleviate the classical issues in symbolic processing. Their vast knowledge can also be utilized to reduce the need for rebuilding neural components, allowing for flexible connections with different symbolic components.

10 Conclusion

NeSy AI presents a promising path forward in addressing the limitations of purely symbolic or neural approaches to AI. By integrating symbolic reasoning with neural learning, NeSy frameworks offer a balance between interpretability, data usage, time efficiency, and generalization. In this paper, we characterize the core components of NeSy frameworks and provide an analysis of some existing ones—DeepProbLog, Scallop, and DomiKnowS, illustrating their comparative facets. We identified a number of facets, such as symbolic knowledge and data representation, neural modeling, model declaration, their method of integrating the symbolic and sub-symbolic systems, and LLMs integration. We identify key challenges in each facet that can guide us toward building the next generation of NeSy frameworks. Unifying ideas in the field and building flexible frameworks by incorporating strengths in every facet will ease the learning curve associated with NeSy systems and improve their application. Future NeSy frameworks should aim to provide user-friendly interfaces, scalability, algorithm coverage, and seamless integrations with foundation models. Recent advances in LLMs and vision language models provide promising solutions to longstanding knowledge engineering challenges, fostering more effective and scalable integration of symbolic representations, and advancing research in NeSy AI.

Acknowledgement

Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Office of Naval Research. The authors thank Uzair Mohammad for his editing and suggestions for Figure 1.

Funding

The authors disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This project is partially supported by the Office of Naval Research (ONR—grant N00014-23-1-2417).

Declaration of Conflicting Interests

The authors declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

ORCID iDs

Tanawan Premsri https://orcid.org/0009-0004-0223-860X
Danial Kamali https://orcid.org/0000-0002-2652-2339
Parisa Kordjamshidi https://orcid.org/0000-0002-4606-1824

Note

1 https://github.com/HLR/nesy-examples

References

Abiteboul, S., Hull, R., & Vianu, V. (1995). *Foundations of databases: The logical level 1st Ed*. USA: Addison-Wesley Longman Publishing Co., Inc.

Acharya, K., Velasquez, A., & Song, H. H. (2024). A survey on symbolic knowledge distillation of large language models. *IEEE Transactions on Artificial Intelligence*, *5*(12), 5928–5948. https://doi.org/10.1109/TAI.2024.3428519

Aditya, D., Mukherji, K., Balasubramanian, S., Chaudhary, A., & Shakarian, P. (2023). Pyreason: Software for open world temporal logic. https://arxiv.org/abs/2302.13482.

Ahmad, J., Farman, H., & Jan, Z. (2019). *Deep learning methods and applications*. Singapore: Springer Singapore. https://doi.org/10.1007/978-981-13-3459-7_3

Ahmed, K., Li, T., Ton, T., Guo, Q., Chang, K. W., Kordjamshidi, P., Srikumar, V., Van den Broeck, G., & Singh, S. (2022). Pylon: A pytorch framework for learning with constraints. In D. Kiela, M. Ciccone & B. Caputo (Eds.), *Proceedings of the NeurIPS 2021 competitions and demonstrations track*, *Proceedings of Machine Learning Research*, (Vol. 176, pp. 319–324). PMLR. https://proceedings.mlr.press/v176/ahmed22a.html.

Augusto, L. M. (2021). From symbols to knowledge systems: A. Newell and H. A. Simon's contribution to symbolic AI. *Journal of Knowledge Structures and Systems*, *2*(1), 29–62.

Badreddine, S., d'Avila Garcez, A., Serafini, L., & Spranger, M. (2022). Logic tensor networks. *Artificial Intelligence*, *303*, 103649. https://doi.org/10.1016/j.artint.2021.103649

Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*, FAccT '21 (p. 610–623). New York, NY, USA: Association for Computing Machinery. ISBN 9781450383097. https://doi.org/10.1145/3442188.3445922

Bhuyan, B. P., Ramdane-Cherif, A., Tomar, R., & Singh, T. P. (2024). Neuro-symbolic artificial intelligence: A survey. *Neural Computing and Applications*, *36*(21), 12809–12844. https://doi.org/10.1007/s00521-024-09960-z

Booch, G., Fabiano, F., Horesh, L., Kate, K., Lenchner, J., Linck, N., Loreggia, A., Murgesan, K., Mattei, N., Rossi, F., & Srivastava, B. (2021). Thinking fast and slow in AI. *Proceedings of the AAAI Conference on Artificial Intelligence*, *35*(17), 15042–15046. https://doi.org/10.1609/aaai.v35i17.17765

Bouneffouf, D., & Aggarwal, C. C. (2022). Survey on applications of neurosymbolic artificial intelligence. https://arxiv.org/abs/2209.12618.

Bratko, I., & Muggleton, S. (1995). Applications of inductive logic programming. *Communications of the ACM*, *38*(11), 65–70. https://doi.org/10.1145/219717.219771

Burattini, E., de Francesco, A., & De Gregorio, M. (2002). Nsl: a neuro-symbolic language for monotonic and non-monotonic logical inferences. In *VII Brazilian symposium on neural networks, 2002. SBRN 2002. Proceedings*. (pp. 256–261). https://doi.org/10.1109/SBRN.2002.1181487

Clocksin, W. F., & Mellish, C. S. (2003). *Programming in PROLOG*. Springer Science & Business Media.

Cohen, W. W., Yang, F., & Mazaitis, K. R. (2017). Tensorlog: Deep learning meets probabilistic dbs. *arXiv preprint arXiv:1707.05390*.

Cropper, A., & Dumančić, S. (2022). Inductive logic programming at 30: A new introduction. *Journal of Artificial Intelligence Research*, *74*, 765–850. https://doi.org/10.1613/jair.1.13507

Darwiche, A. (2011). Sdd: A new canonical representation of propositional knowledge bases. In *IJCAI Proceedings-international joint conference on artificial intelligence*, (Vol. 22, p. 819).

De Raedt, L., Kimmig, A., & Toivonen, H. (2007). Problog: a probabilistic prolog and its application in link discovery. In *Proceedings of the 20th international joint conference on artifical intelligence*, IJCAI'07 (pp. 2468–2473). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Derkinderen, V., Manhaeve, R., Adriaensen, R., Praet, L. V., Smet, L. D., Marra, G., & Raedt, L. D. (2025). The deeplog neurosymbolic machine. https://arxiv.org/abs/2508.13697.

Eisner, J. (2002). Parameter estimation for probabilistic finite-state transducers. In *Proceedings of the 40th annual meeting of the association for computational linguistics*. (pp. 1–8).

Fabiano, F., Pallagani, V., Ganapini, M. B., Horesh, L., Loreggia, A., Murugesan, K., Rossi, F., & Srivastava, B. (2023). Plan-SOFAI: A neuro-symbolic planning architecture. In *Neuro-Symbolic learning and reasoning in the era of large language models*. https://openreview.net/forum?id=ORAhay0H4x.

Faghihi, H. R., Nafar, A., Uszok, A., Karimian, H., & Kordjamshidi, P. (2024). Prompt2demodel: Declarative neuro-symbolic modeling with natural language. https://arxiv.org/abs/2407.20513.

Faghihi, H. R., Nafar, A., Zheng, C., Mirzaee, R., Zhang, Y., Uszok, A., Wan, A., Premsri, T., Roth, D., & Kordjamshidi, P. (2023). Gluecons: A generic benchmark for learning under constraints. https://arxiv.org/abs/2302.10914.

Fang, C., & Yu, S. C. (2024). Large language models are neurosymbolic reasoners. *arXiv preprint arXiv:2401.09334* https://arxiv.org/html/2401.09334v1.

Frazier, M., & Pitt, L. (1993). Learning from entailment: An application to propositional horn sentences. In *Proceedings of the tenth international conference on international conference on machine learning*. (pp. 120–127).

Giunchiglia, E., Tatomir, A., Stoian, M. C., & Lukasiewicz, T. (2024). Ccn+: A neuro-symbolic framework for deep learning with requirements. *International Journal of Approximate Reasoning*, *171*, 109124. https://doi.org/10.1016/j.ijar.2024.109124

Green, T. J., Karvounarakis, G., & Tannen, V. (2007). Provenance semirings. In: *Proceedings of the twenty-sixth acm sigmod-sigact-sigart symposium on principles of database systems*, PODS '07 (p. 31–40). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/1265530.1265535

Guo, Q., Rajaby Faghihi, H., Zhang, Y., Uszok, A., & Kordjamshidi, P. (2020). Inference-masked loss for deep structured output learning. In C. Bessiere (Ed.), *Proceedings of the twenty-ninth international joint conference on artificial intelligence, IJCAI-20* (pp. 2754–2761). International Joint Conferences on Artificial Intelligence Organization. https://doi.org/10.24963/ijcai.2020/382

Gurobi Optimization, LLC. (2024) Gurobi Optimizer Reference Manual. https://www.gurobi.com.

Hayes-Roth, F. (1985). Rule-based systems. *Communications of the ACM*, *28*(9), 921–932.

Hitzler, P., & Sarker, M. K. (2021). *Neuro-symbolic artificial intelligence: The state of the Art*, *Frontiers in artificial intelligence and applications*, (Vol. 342). IOS Press. https://doi.org/10.3233/FAIA342

Hossain, D., & Chen, J. Y. (2025). A study on neuro-symbolic artificial intelligence: Healthcare perspectives. https://arxiv.org/abs/2503.18213.

Hsu, J., Mao, J., Tenenbaum, J. B., & Wu, J. (2023). What's left? concept grounding with logic-enhanced foundation models. https://arxiv.org/abs/2310.16035.

Huang, J., Li, Z., Chen, B., Samel, K., Naik, M., Song, L., & Si, X. (2021). Scallop: From probabilistic deductive databases to scalable differentiable reasoning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang & J. W. Vaughan (Eds.), *Advances in neural information processing systems*, (Vol. 34, pp. 25134–25145). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2021/file/d367eef13f90793bd8121e2f675f0dc2-Paper.pdf.

Ishay, A., Yang, Z., & Lee, J. (2023). Leveraging large language models to generate answer set programs. In S. Marquis, T. C. Son & G. Kern-Isberner (Eds.), *Proceedings of the 20th international conference on principles of knowledge representation and reasoning, KR 2023*, Proceedings of the International Conference on Knowledge Representation and Reasoning. Association for the Advancement of Artificial Intelligence, (pp. 374–383). https://doi.org/10.24963/kr.2023/37

Jayasingha, P., Iancu, B., & Lilius, J. (2025). Neurosymbolic approaches in AI design – an overview. In *2025 IEEE symposium on trustworthy, explainable and responsible computational intelligence (CITREx Companion)*. (pp. 1–5). https://doi.org/10.1109/CITRExCompanion65208.2025.10981497

Johnson, J., Hariharan, B., der Maaten, L., van Fei-Fei, L., Lawrence Zitnick, C., & Girshick, R. (2017). Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*.

Kahneman, D. (2011). *Thinking, Fast and Slow*. macmillan.

Kamali, D., Barezi, E. J., & Kordjamshidi, P. (2025). Nesycoco: A neuro-symbolic concept composer for compositional generalization. *Proceedings of the AAAI Conference on Artificial Intelligence*, *39*(4), 4184–4193. https://doi.org/10.1609/aaai.v39i4.32439

Kautz, H. (2022). The third AI summer: AAAI Robert S. Engelmore memorial lecture. *AI Magazine*, *43*(1), 105–125. https://doi.org/10.1002/aaai.12036

Kimmig, A., Van den Broeck, G., & De Raedt, L. (2011). An algebraic prolog for reasoning about possible worlds. *Proceedings of the AAAI Conference on Artificial Intelligence*, *25*(1), 209–214. https://doi.org/10.1609/aaai.v25i1.7852

Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A. C., Lo, W. Y., Dollár, P., & Girshick, R. (2023). Segment anything. https://arxiv.org/abs/2304.02643.

Kolaitis, P. G., & Vardi, M. Y. (1990). On the expressive power of datalog: tools and a case study. In *Proceedings of the Ninth ACM sigact-sigmod-sigart symposium on principles of database systems*, PODS '90 (p. 61–71). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/298514.298542

Kordjamshidi, P., Khashabi, D., Christodoulopoulos, C., Mangipudi, B., Singh, S., & Roth, D. (2016). Better call Saul: Flexible programming for learning and inference in NLP. In Y. Matsumoto & R. Prasad (Eds.), *Proceedings of COLING 2016, the 26th international conference on computational linguistics: Technical papers* (pp. 3030–3040). Osaka, Japan: The COLING 2016 Organizing Committee. https://aclanthology.org/C16-1285/.

Kordjamshidi, P., Roth, D., & Kersting, K. (2022). Declarative learning-based programming as an interface to AI systems. *Frontiers in Artificial Intelligence*, *5*(2022), 755361. https://doi.org/10.3389/frai.2022.755361

Kordjamshidi, P., Roth, D., & Wu, H. (2015). Saul: Towards declarative learning based programming. In *AAAI Fall Symposia*. (p. 12).

Lamb, L. C., d'Avila Garcez, A. S., Gori, M., Prates, M. O. R., Avelar, P. H. C., & Vardi, M. Y. (2020). Graph neural networks meet neural-symbolic computing: A survey and perspective. In C. Bessiere (Ed.), *Proceedings of the twenty-ninth international joint conference on artificial intelligence, IJCAI 2020* (pp. 4877–4884). ijcai.org. https://doi.org/10.24963/ijcai.2020/679

Lample, G., & Charton, F. (2020). Deep learning for symbolic mathematics. In *International Conference on Learning Representations*. https://openreview.net/forum?id=S1eZYeHFDS.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. https://doi.org/10.1109/5.726791

Li, B., Qi, P., Liu, B., Di, S., Liu, J., Pei, J., Yi, J., & Zhou, B. (2023a). Trustworthy AI: From principles to practices. *ACM Computing Surveys*, *55*(9), 177:1–177:46. https://doi.org/10.1145/3555803

Li, T., & Srikumar, V. (2019). Augmenting neural networks with first-order logic. In A. Korhonen, D. Traum & L. Màrquez (Eds.), *Proceedings of the 57th annual meeting of the association for computational linguistics* (pp. 292–302). Florence, Italy: Association for Computational Linguistics. https://doi.org/10.18653/v1/P19-1028

Li, Z., Huang, J., Liu, J., Zhu, F., Zhao, E., Dodds, W., Velingker, N., Alur, R., & Naik, M. (2024). Relational programming with foundational models. *Proceedings of the AAAI Conference on Artificial Intelligence*, *38*(9), 10635–10644. https://doi.org/10.1609/aaai.v38i9.28934

Li, Z., Huang, J., & Naik, M. (2023b). Scallop: A language for neurosymbolic programming. *Proceedings of the ACM on Programming Languages 7*(PLDI), 166:1–166:25. https://doi.org/10.1145/3591280

Lima, P. M. V., Morveli-Espinoza, M. M., Pereira, G. C., & Franga, F. (2005). Satyrus: a sat-based neuro-symbolic architecture for constraint processing. In *Fifth international conference on hybrid intelligent systems (HIS'05)* (pp. 6–pp). IEEE.

Liu, R., Liu, C., Bai, Y., & Yuille, A. L. (2019). Clevr-ref+: Diagnosing visual reasoning with referring expressions. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*.

Lu, Z., Afridi, I., Kang, H. J., Ruchkin, I., & Zheng, X. (2024). Surveying neuro-symbolic approaches for reliable artificial intelligence of things. *Journal of Reliable Intelligent Environments*, *10*(3), 257–279. https://doi.org/10.1007/s40860-024-00231-1

Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., & De Raedt, L. (2021). Neural probabilistic logic programming in deepproblog. *Artificial Intelligence*, *298*, 103504. https://doi.org/10.1016/j.artint.2021.103504

Mao, J., Gan, C., Kohli, P., Tenenbaum, J. B., & Wu, J. (2019). The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. https://arxiv.org/abs/1904.12584.

Minderer, M., Gritsenko, A. A., Stone, A., Neumann, M., Weissenborn, D., Dosovitskiy, A., Mahendran, A., Arnab, A., Dehghani, M., Shen, Z., Wang, X., Zhai, X., Kipf, T., & Houlsby, N. (2022). Simple open-vocabulary object detection. In S. Avidan, G. J. Brostow, M. Cissé, G. M. Farinella & T. Hassner (Eds.), *Computer Vision - ECCV 2022 - 17th European conference, Tel Aviv, Israel, October 23-27, 2022, Proceedings, Part X*, *Lecture Notes in Computer Science*, (Vol. 13670, pp. 728–755). Springer. https://doi.org/10.1007/978-3-031-20080-9_42

Mirzaee, R., & Kordjamshidi, P. (2023). Disentangling extraction and reasoning in multi-hop spatial reasoning. In H. Bouamor, J. Pino & K. Bali (Eds.), *Findings of the association for computational linguistics: EMNLP 2023* (pp. 3379–3397). Singapore: Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.findings-emnlp.221

Nandwani, Y., Pathak, A., & Mausam Singla, P. (2019). A primal dual formulation for deep learning with constraints. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox & R. Garnett (Eds.), *Advances in neural information processing systems*, (Vol. 32). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2019/file/cf708fc1decf0337aded484f8f4519ae-Paper.pdf.

Newell, A. (1980). Physical symbol systems. *Cognitive Science*, *4*(2), 135–183. https://doi.org/10.1016/S0364-0213(80)80015-2

Ng, R., & Subrahmanian, V. (1992). Probabilistic logic programming. *Information and Computation*, *101*(2), 150–201. https://doi.org/10.1016/0890-5401(92)90061-J

Nienhuys-Cheng, S. H., & de Wolf, R. (1997). *Foundations of Inductive Logic Programming*. Springer.

OpenA, I., Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., & Avila, R. (2024). Gpt-4 technical report. https://arxiv.org/abs/2303.08774.

Pan, L., Albalak, A., Wang, X., & Wang, W. (2023a). Logic-LM: Empowering large language models with symbolic solvers for faithful logical reasoning. In H. Bouamor, J. Pino & K. Bali (Eds.), *Findings of the association for computational linguistics: EMNLP 2023* (pp. 3806–3824). Singapore: Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.findings-emnlp.248

Pan, L., Albalak, A., Wang, X., & Wang, W. Y. (2023b). Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. https://arxiv.org/abs/2305.12295.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., & Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. https://arxiv.org/abs/1912.01703.

Petroni, F., Rocktäschel, T., Lewis, P., Bakhtin, A., Wu, Y., Miller, A. H., & Riedel, S. (2019). Language models as knowledge bases? https://arxiv.org/abs/1909.01066.

Plaat, A., Van Duijn, M., Van Stein, N., Preuss, M., Van der Putten, P., & Batenburg, K. J. (2025). Agentic large language models, a survey. *Journal of Artificial Intelligence Research*, *84*, 29:1–29:74. https://doi.org/10.1613/jair.1.18675

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., & Sutskever, I. (2021). Learning transferable visual models from natural language supervision. https://arxiv.org/abs/2103.00020.

Rajaby Faghihi, H., Guo, Q., Uszok, A., Nafar, A., & Kordjamshidi, P. (2021). DomiKnowS: A library for integration of symbolic domain knowledge in deep learning. In H. Adel & S. Shi (Eds.), *Proceedings of the 2021 conference on empirical methods in natural language processing: System demonstrations* (pp. 231–241). online and Punta Cana, Dominican Republic: Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.emnlp-demo.27

Sathasivam, S. (2011). Learning rules comparison in neuro-symbolicintegration. *International Journal of Applied Physics and Mathematics*, *1*(2), 129.

Serafini, L., & d'Avila Garcez, A. S. (2016). Learning and reasoning with logic tensor networks. In *Conference of the italian association for artificial intelligence*(pp. 334–348). Springer.

Shpilka, A., & Yehudayoff, A. (2010). Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends® in Theoretical Computer Science*, *5*(3–4), 207–388.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489. https://doi.org/10.1038/nature16961

Smolensky, P., Lee, M., He, X., Yih, W. T., Gao, J., & Deng, L (2016). Basic reasoning with tensor product representations. https://doi.org/10.48550/arXiv.1601.02745

Sun, J., Sun, H., Han, T., & Zhou, B. (2021). Neuro-symbolic program search for autonomous driving decision module design. In J. Kober, F. Ramos & C. Tomlin (Eds.) *Proceedings of the 2020 conference on robot learning*, *Proceedings of Machine Learning Research*, (Vol. 155, pp. 21–30). PMLR. https://proceedings.mlr.press/v155/sun21a.html.

Tjong Kim Sang, E. F., & De Meulder, F. (2003). Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh conference on natural language learning at HLT-NAACL 2003* (pp. 142–147). https://aclanthology.org/W03-0419/.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., & Scialom, T. (2023). Llama 2: Open foundation and fine-tuned chat models. https://arxiv.org/abs/2307.09288.

Van Hentenryck, P., Simonis, H., & Dincbas, M. (1992). Constraint satisfaction using constraint logic programming. *Artificial Intelligence*, *58*(1), 113–159. https://doi.org/10.1016/0004-3702(92)90006-J

Weizenbaum, J. (1966). Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, *9*(1), 36–45. https://doi.org/10.1145/365153.365168

Wong, L., Grand, G., Lew, A. K., Goodman, N. D., Mansinghka, V. K., Andreas, J., & Tenenbaum, J. B. (2023). From word models to world models: Translating from natural language to the probabilistic language of thought. https://arxiv.org/abs/2306.12672.

Xu, F., Wu, Z., Sun, Q., Ren, S., Yuan, F., Yuan, S., Lin, Q., Qiao, Y., & Liu, J. (2024a). Symbol-LLM: Towards foundational symbol-centric interface for large language models. In L. W. Ku, A. Martins & V. Srikumar (Eds.), *Proceedings of the 62nd Annual meeting of the association for computational linguistics (Volume 1: Long Papers)* (pp. 13091–13116). Bangkok, Thailand: Association for Computational Linguistics. https://doi.org/10.18653/v1/2024.acl-long.707

Xu, J., Zhang, Z., Friedman, T., Liang, Y., & Van den Broeck, G. (2018). A semantic loss function for deep learning with symbolic knowledge. In J. Dy & A. Krause (Eds.), *Proceedings of the 35th international conference on machine learning*, *Proceedings of Machine Learning Research*, (Vol. 80, pp. 5502–5511). PMLR. https://proceedings.mlr.press/v80/xu18h.html.

Xu, X., Wang, P., Dong, S., Wu, N., Zhang, Y., & Chang, B. (2024b). Faithful logical reasoning via symbolic chain-of-thought. *arXiv preprint arXiv:2405.18357* https://arxiv.org/abs/2405.18357.

Yang, X., Chen, B., & Tam, Y. C. (2024). Arithmetic reasoning with LLM: Prolog generation & permutation. In K. Duh, H. Gomez & S. Bethard (Eds.), *Proceedings of the 2024 conference of the north american chapter of the association for computational linguistics: Human language technologies (Volume 2: Short Papers)* (pp. 699–710). Mexico City, Mexico: Association for Computational Linguistics. https://doi.org/10.18653/v1/2024.naacl-short.61

Yao, L., Peng, J., Mao, C., & Luo, Y. (2025). Exploring large language models for knowledge graph completion. https://arxiv.org/abs/2308.13916.

Yi, K., Wu, J., Gan, C., Torralba, A., Kohli, P., & Tenenbaum, J. B. (2018). Neural-symbolic VQA: Disentangling reasoning from vision and language understanding. In *Advances in neural information processing systems*. (pp. 1039–1050).

Zhang, H., Huang, J., Li, Z., Naik, M., & Xing, E. (2023). Improved logical reasoning of language models via differentiable symbolic programming. https://arxiv.org/abs/2305.03742.

Zhang, H., Kung, P. N., Yoshida, M., den Broeck, G. V., & Peng, N. (2024). Adaptable logical control for large language models. In *The Thirty-eighth annual conference on neural information processing systems*. https://openreview.net/forum?id=58X9v92zRd.

Zheng, D., Lapata, M., & Pan, J. Z. (2024). How reliable are llms as knowledge bases? re-thinking facutality and consistency. https://arxiv.org/abs/2407.13578.