An Overview of Neuro-Symbolic Frameworks for Software Development

Luca Ferrante[1], Stefano Forti[1,*] and Jacopo Soldani[1]

[1]Department of Computer Science, University of Pisa, Italy

Abstract

Neuro-symbolic artificial intelligence aims at combining the learning capabilities of neural models with the structured reasoning provided by symbolic representations. Despite a growing number of frameworks supporting this integration, from a developer's perspective, the design space of neuro-symbolic systems remains fragmented, making it difficult to compare approaches systematically and to understand their underlying trade-offs. In this article, we present a systematic literature review and an original taxonomy-driven analysis of open-source development frameworks for building neuro-symbolic software systems. We consider four orthogonal dimensions: (*i*) linguistic aspects, capturing the underlying logic formalism and expressiveness, (*ii*) inference and reasoning mechanisms, including semantics, solvers, and differentiability, (*iii*) neuro-symbolic integration pipelines, interpreted through Kautz's architectural patterns, and (*iv*) usability and engineering considerations. The analysis identifies open challenges related to balancing expressiveness and scalability, improving engineering maturity, and performing comparisons over standard benchmarks.

Keywords

Neuro-symbolic artificial intelligence, Logic programming, Systematic literature review, Software engineering

1. Introduction

The relation between learning and reasoning has shaped artificial intelligence (AI) since its earliest formulations. On one hand, symbolic AI made knowledge explicit through logic, rules, ontologies, and declarative programmes, thereby supporting explanation, compositionality, and reasoning under constraints. On the other hand, neural AI enabled robust pattern recognition from high-dimensional data, but often at the cost of opacity in decision-making, data hunger, and limited out-of-distribution generalisation. Neuro-symbolic AI revisits this long-standing divide by seeking how to suitably and fruitfully combine learned representations with symbolic knowledge and inference [54, 44, 55].

When considering software development frameworks, neuro-symbolic AI is not merely an algorithmic question. Indeed, a framework fixes a modelling language, a semantic interpretation of symbolic knowledge, a solver or relaxation strategy, a training loop, and practical assumptions, e.g. about data, hardware, and developer expertise. Said otherwise, development frameworks shape what can be expressed, what can be learned, what can be explained, and what can be deployed, and, as developers, it is important to fully understand these aspects to make informed decisions for a given software project.

Existing surveys and literature reviews have mapped the broader neuro-symbolic landscape from various complementary viewpoints. Some efforts focus on the relation between statistical relational learning and neuro-symbolic AI [49], others organise recent work by research area, application domain, or high-level architecture [43, 50, 48, 45]. These contributions leave a specific gap for researchers and practitioners who need to choose the best framework(s) to develop neuro-symbolic software systems. Our work aims at complementing previous efforts by focussing on reusable open-source development frameworks and comparing them through developer-facing commitments. Indeed, to take their decisions, developers need to understand the exposed symbolic language, the used reasoning substrate and neuro-symbolic integration pattern, and its engineering support and maturity level.

CILC 2026: The 41st Italian Conference on Computational Logic, June 23–25, 2026, Ferrara, Italy

*Corresponding author.

Author names are alphabetically sorted; all Authors have contributed equally to this study.

l.ferrante8@studenti.unipi.it (L. Ferrante); stefano.forti@unipi.it (S. Forti); jacopo.soldani@unipi.it (J. Soldani)

https://pages.di.unipi.it/forti (S. Forti); https://pages.di.unipi.it/soldani (J. Soldani)

Through a systematic literature review (SLR) of the state of the art, this article addresses the identified gap by analysing open-source neuro-symbolic development frameworks through an original taxonomy. After briefly reviewing the state of the art, we focus on a selection of the available frameworks, which corresponds to the best documented among the general-purpose and open-source ones.

Our contribution is threefold. First, we propose a novel four-dimensional taxonomy covering frameworks' linguistic aspects, inference and reasoning, neuro-symbolic pipelines, and usability. Second, we apply this taxonomy to analyse a selection of general-purpose frameworks, from probabilistic logic programming and Answer Set Programming (ASP) extensions to fuzzy first-order systems, constraint-guided neural learning, and probabilistic programming. Third, as a result of our taxonomy-driven analyses, we identify some open challenges in the field.

Overall, our systematic literature review is guided by three research questions:

RQ1. *Which open-source frameworks support neuro-symbolic software system development, and how can they be positioned by scope and application domain?*

RQ2. *Which linguistic, reasoning, integration, and engineering aspects distinguish available general-purpose and open-source frameworks?*

RQ3. *Which limitations and open challenges characterise the current framework landscape?*

The rest of this article is organised as follows. Sect. 2 describes the systematic review protocol, literature selection process, and threats to validity. Sect. 3 reviews the framework corpus and addresses **RQ1**. Sect. 4 applies the taxonomy to the selected frameworks and addresses **RQ2**. Sect. 5 discusses identified open challenges, addressing **RQ3**. Sect. 6 briefly concludes the article.

2. Scope and Methodology

Our work is grounded in an SLR of peer-reviewed literature on tools, libraries, and frameworks for developing neuro-symbolic systems. The protocol followed the usual sequence of database search, duplicate removal, title-and-abstract screening, full-text screening, data extraction, and backward snowballing, as sketched in Fig. 1. We followed the systematic mapping study guidelines proposed by Petersen et al. [51, 52], adapting them to the objectives of our work.

Search strategy and corpus selection. The search string was derived from PICO (*Population*, *Intervention*, *Context*, *Outcome*) terms characterising our research questions. In detail, population terms denoted neuro-symbolic software systems and development frameworks. Intervention and phenomenon terms denoted tools, libraries, frameworks, software artefacts, platforms, and open-source implementations. Context terms referred to artificial intelligence and software development, while outcome terms targeted reusable evidence for framework identification, classification, and comparison. The search covered major scientific indexes, including Scopus, Web of Science, ACM Digital Library, and IEEE Xplore, using the following search string, adapted to each database syntax:

$$\texttt{neuro symbolic} \wedge (\texttt{artificial intelligence} \vee \texttt{AI}) \wedge$$
$$(\texttt{approach*} \vee \texttt{tool*} \vee \texttt{library} \vee \texttt{framework} \vee$$
$$\texttt{implementation} \vee \texttt{software} \vee \texttt{platform*}) \wedge \texttt{open-source}$$

The database search was run in February 2026. Articles indexed after that date are outside the scope of this work. To identify the articles in the corpus, we applied a specific set of inclusion and exclusion criteria. Inclusion criteria required a primary research contribution, a clear neuro-symbolic component, and an open-source software artefact or implementation-oriented framework. Exclusion criteria removed secondary studies, unavailable full texts, articles without a reusable software component, works not focussing on neuro-symbolic integration, and non-peer-reviewed material and collections.

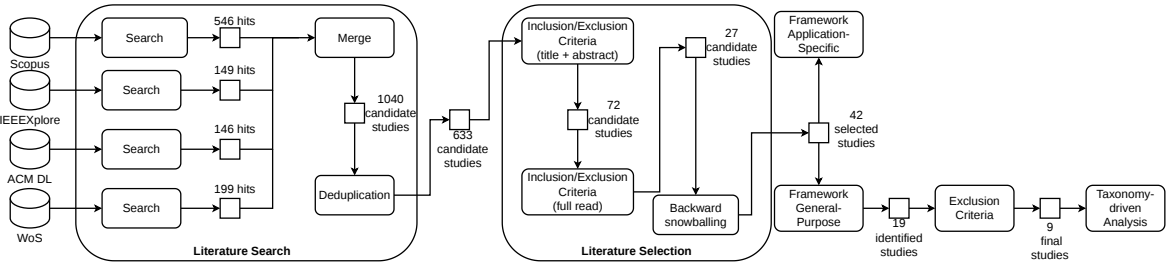

Figure 1: SLR process for identifying and classifying existing literature of interest.

Screening was conducted in two passes. The first pass filtered records by title and abstract. The second pass inspected full texts against the inclusion criteria. Before conflict resolution, the first pass involved 633 records with an inter-rater agreement of 80.6%.[1] The second pass involved 72 full texts, with an inter-rater agreement of 84.7%. Disagreements were resolved through discussion and, when needed, through an additional reviewer. Eventually, the process retained 42 corpus articles published between 2017 and February 2026, corresponding to 41 frameworks. All selected articles are listed in Table 1.

Table 1
Corpus of SLR studies.

Ref.	System or study	Year	Scope
[1]	LDM: FAIR and federated research-data management	2025	Application-specific
[2]	FLPN: zero-shot learning through neuro-symbolic integration	2026	Application-specific
[3]	NeSyCoCo: concept composition for compositional generalisation	2025	Application-specific
[4]	BioLinkerAI: biomedical entity linking and knowledge capture	2025	Application-specific
[5]	dNL-NL: explanatory rules in non-linear domains	2024	General-purpose
[6]	Prompt2DeModel: natural-language declarative modelling	2024	Application-specific
[7]	RevelioNN: ontology-based prediction and explainability	2024	Application-specific
[8]	Neural networks for abstraction and reasoning	2024	Application-specific
[9]	NeuroSynt: portfolio solving for reactive synthesis	2024	Application-specific
[10]	Chebifier: semantic classification in ChEBI	2024	Application-specific
[11]	Relational programming with foundation models	2024	General-purpose
[12]	CCN+: deep learning with requirements	2024	General-purpose
[13]	ULLER: unified language for learning and reasoning	2024	General-purpose
[14]	Embed2Rule: latent-space weak labelling for rule learning	2024	General-purpose
[15]	Logic Tensor Probe: probing LLMs for logical reasoning	2024	Application-specific
[16]	Scallop: language for neuro-symbolic programming	2023	General-purpose
[17]	NSIL: learning answer-set programmes from raw data	2023	General-purpose
[18]	PyReason: open-world temporal logic software	2023	General-purpose
[19]	Faster-LTN: end-to-end object detection	2021	Application-specific
[20]	SLASH: scalable neural-probabilistic ASP	2023	General-purpose
[21]	Composite AI with compact neuro-symbolic models	2025	Application-specific
[22]	Neuro-symbolic travel-demand prediction	2025	Application-specific
[23]	TinyNS: platform-aware TinyML	2024	Application-specific
[24]	DANLI: deliberative instruction-following agent	2022	Application-specific
[25]	KnowZRel: zero-shot relationship retrieval	2025	Application-specific
[26]	Nessy: label-noise reduction	2022	Application-specific
[27]	LTN: logic tensor networks	2022	General-purpose
[28]	Knowledge-grounded planning and reasoning systems	2025	Application-specific
[29]	αILP: differentiable logic programmes for visual scenes	2023	General-purpose
[30]	Neural-probabilistic answer-set programming	2022	General-purpose
[31]	DeepProbLog: neural probabilistic logic programming	2021	General-purpose
[32]	DeepProbLogCEP: complex event processing over audio streams	2021	Application-specific
[33]	PYLON: PyTorch framework for learning with constraints	2022	General-purpose
[34]	LTNtorch: PyTorch implementation of logic tensor networks	2025	General-purpose
[35]	PROTO-LTN: prototypical logic tensor networks	2022	Application-specific
[36]	Edward: deep probabilistic programming	2017	General-purpose
[37]	NeurASP: neural networks in ASP	2020	General-purpose
[38]	Neuro-symbolic ASP pipeline for visual question answering	2022	Application-specific
[39]	DeepStochLog: neural stochastic logic programming	2022	General-purpose
[40]	NeuPSL: neural probabilistic soft logic	2022	General-purpose
[41]	DomiKnowS: symbolic domain knowledge in deep learning	2021	General-purpose
[42]	FFNSL: feed-forward neural-symbolic learner	2023	General-purpose

Table 2 summarises the temporal distribution of the collected articles and distinguishes application-specific studies from general-purpose framework contributions per year[2]. Most contributions are concentrated in recent years, rising from 2021 onward with a peak in 2024, and reflecting the increasing availability of open-source neuro-symbolic frameworks in the application-specific domain.

Corpus classification. The selected corpus contains both general-purpose and application-specific frameworks. On one hand, general-purpose frameworks expose reusable languages, APIs, solvers, or

[1]We measured the inter-rater agreement on the inclusion/exclusion coding by exploiting the Krippendorff-α coefficient [47], which classifies a coding as good if the value is greater or equal to 80%.

[2]Table 2 reports corpus articles by year and scope. The subsequent count of 41 framework-level approaches collapses articles that refer to the same framework family, yielding 22 application-specific and 19 general-purpose frameworks.

Table 2
Distribution of corpus articles by publication year and framework scope.

Year	Application-specific	General-purpose	Total
2017	0	1	1
2020	0	1	1
2021	2	2	4
2022	4	5	9
2023	0	6	6
2024	7	5	12
2025	7	1	8
2026	1	0	1
Total	**21**	**21**	**42**

training mechanisms. Conversely, application-specific systems show how neuro-symbolic ideas are being operationalised in specific domains, such as knowledge-graph management, visual reasoning, entity linking, reactive synthesis, or TinyML. After a first analysis of this corpus, we further restrict our study to consider the 19 general-purpose frameworks and, among these, only those 9 that feature suitable documentation, thus allowing developers to use them in their software projects. We considered documentation suitable when it allowed an independent developer to understand the modelling language, obtain and install the framework, and run at least one example.

After frameworks identification, the collected data was organised through a coding schema aligned with the goal of comparing the selected general-purpose neuro-symbolic software frameworks from a developer's viewpoint. The schema was tested and refined on the selected frameworks. The resulting taxonomy is organised into four dimensions (i.e. linguistic aspects, inference & reasoning, neuro-symbolic pipeline, and usability & engineering), which are applied in Sect. 4.

Replication package. A replication package is released as an online spreadsheet.[3] It documents the protocol, search results before and after duplicate removal, inclusion/exclusion decisions, snowballing, the final corpus, and data used for the taxonomy-driven analysis. For each retained framework, we recorded repository metadata, license information, and last-maintenance evidence (i.e. last commit) as indicators of engineering maturity.

Threats to validity. Naturally, our work has been subject to threats to validity, which we have tried to mitigate and briefly discuss hereinafter as per the standard taxonomy proposed by Wohlin et al [56].

Construct validity is affected by the fact that categories such as "framework", "tool", and "application-specific system" are not always sharply separated in the neuro-symbolic literature. While building our corpus, we tried to make the query string as inclusive as possible by extending it with synonyms and alternative spellings. Besides, the proposed taxonomy mitigates this risk by classifying frameworks through observable design commitments rather than by labels used by the authors of individual articles. Internal validity concerns arise from screening and coding decisions, especially when an article reports both a method and a software artefact. These decisions were checked against the extracted review material and revised by all authors when conflicts arose.

External validity is limited by the rapid evolution of open-source neuro-symbolic software. Repository availability, documentation, dependencies, and maintenance status may change after publication. Our work therefore treats engineering maturity as time-sensitive evidence rather than as a permanent property. Last, conclusion validity is limited by the heterogeneity of tasks and evaluation protocols. To this end, we focus on comparative claims about languages, semantics, reasoning mechanisms, integration patterns, and engineering aspects, as per the proposed taxonomy. The actual comparison of selected frameworks against specific benchmarks is currently left for future work.

[3] https://docs.google.com/spreadsheets/d/1FDhApZAdCQRN8439wNBC2xpiRynM01W0bLOMlfVg4Y4/edit?usp=sharing

3. Neuro-Symbolic Frameworks for Software Development

In this section, the identified corpus is organised into two main categories, viz. application-specific and general-purpose frameworks, as reported in Table 1. Application-specific frameworks target specific tasks or domains through a neuro-symbolic approach, whereas general-purpose frameworks provide reusable languages, APIs, solvers, or modelling mechanisms that can be employed across multiple applications. We offer a concise overview of both categories, with the goal of positioning the identified systems according to their scope and selecting the general-purpose frameworks that will be subject to the taxonomy-based analyses presented later in Sect. 4.

3.1. Application-Specific Systems

Application-specific systems use neuro-symbolic integration to address concrete (type of) datasets or operational settings. We group application-specific systems by the role played by neuro-symbolic integration, i.e. knowledge-centric mediation, visual or planning-oriented reasoning, operational constraints, and adaptations of general-purpose frameworks.

Several systems are knowledge-centric. LDM [1] combines knowledge graphs, entity linking, LLM-based interaction, and federated query processing for FAIR-oriented management of research digital objects such as datasets, publications, and software. BioLinkerAI [4] applies similar knowledge-based mediation to biomedical entity linking, aiming at extracting biomedical entities from text and selecting suitable mapping candidates through an LLM-mediated step. Chebifier [10] uses semantic classification for compound classification from chemical representations, while RevelioNN [7] connects ontology-based modelling with neural prediction and explainability for PyTorch binary classifiers.

Other systems focus on visual, multimodal, or planning-oriented tasks. NeSyCoCo [3] uses LLM-generated symbolic representations and differentiable composition for visual-linguistic compositional reasoning over image-query pairs. DreamCoder [8] tackle program synthesis in the Abstraction and Reasoning Corpus (ARC) through neural-guided search over analysed target programmes. The neuro-symbolic ASP VQA pipeline [38] combines neural perception with answer-set reasoning by mapping visual predictions into logical programs for answering queries. KnowZRel [25] uses common-sense knowledge for zero-shot relationship retrieval in generalised scene-graph generation, while DANLI [24] combines neural language understanding with symbolic deliberation for embodied natural-language instruction following. Similarly, knowledge-grounded planning systems [28] combine foundation-model interaction, knowledge graphs, reasoners, and constraint-aware planning.

A third group is defined by specialised operational constraints. NeuroSynt [9] uses neural guidance for reactive synthesis from temporal-logic specifications, and DeepProbLogCEP [32] applies probabilistic logic programming to complex event processing over audio streams. Nessy [26] addresses label-noise reduction, TinyNS [23] targets platform-aware TinyML code generation for microcontrollers, and rule-guided travel-demand prediction [22] integrates decision-tree rules into neural prediction over socio-economic and mobility data. Composite enterprise AI systems [21] similarly illustrate task-oriented combinations of compact neural and symbolic components for diet management, industrial-production copiloting, and student mental-health support.

Last, some application-specific systems rely on general-purpose frameworks (presented next) by adapting them to narrower domains. Faster-LTN [19], PROTO-LTN [35], FLPN [2], and the Logic Tensor Probe [15] apply LTN-style modelling to semantic image interpretation and object detection, few-shot and zero-shot learning, zero-shot image recognition through visual-semantic prototype matching, or LLM logical-reasoning diagnostics. Prompt2DeModel [6] generates DomiKnowS-style declarative domain models from natural-language prompts, and relational programming with foundation models [11] extends relational specifications with calls to foundation models.

3.2. General-Purpose Frameworks

General-purpose frameworks support the development of software systems across various domains. We organise general-purpose frameworks by the modelling support they expose, ranging from constraints

and fuzzy logic to probabilistic logic, ASP-bases reasoning, rule learning, and probabilistic programming.

First, constraint-based frameworks expose symbolic knowledge as constraints over neural outputs. PYLON [33] lets developers define Python constraints that can be compiled into losses for PyTorch models. LTN [27] and LTNtorch [34] provide fuzzy first-order logic over tensor groundings, while DomiKnowS [41] represents concepts, relations, and constraints as structured domain knowledge connected to neural sensors. CCN+ [12] and NeuPSL [40] also belong to this constraint-oriented area, allowing users to specify requirements for neural models and predicates in probabilistic logic.

Fully probabilistic logic and rule-based frameworks make symbolic programmes more explicit. DeepProbLog [31] extends ProbLog with neural predicates, and DeepStochLog [39] uses stochastic definite clause grammars for sequential reasoning. Neural-probabilistic ASP [30] and its successor SLASH [20] connect neural predictions to answer-set programming with probabilistic predicates, while NeurASP [37] embeds neural atoms in ASP programmes. Scallop [16] offers a Datalog-like relational language with differentiable provenance. These systems expose reusable symbolic languages or reasoning mechanisms, although their semantics and inference strategies differ.

Other general-purpose or implementation-oriented systems emphasise rule learning, temporal reasoning, or broader programming abstractions. PyReason [18] supports open-world temporal reasoning over structured data. NSIL [17], FFNSL [42], αILP [29], Embed2Rule [14], and dNL-NL [5] focus on learning symbolic rules or programmes from data. ULLER [13] proposes a unified language perspective for learning and reasoning. Finally, Edward [36] and Edward2 [53] provide probabilistic-programming abstractions based on random variables and TensorFlow/Python programs, making them especially relevant when uncertainty is to be modelled.

The distinction between application-specific systems and general-purpose frameworks motivates the taxonomy-driven analysis that follows. The in-depth qualitative analysis, presented in the next section, focusses on nine proposals (Table 3) selected from the broader set of general-purpose frameworks identified above. Those frameworks were retained because they provide open-source implementation artefacts, expose modelling mechanisms that are not tied to a single application scenario, and include sufficient documentation or technical detail to support our analyses. The subset also preserves representativeness across different neuro-symbolic approaches, including constraint-guided learning, fuzzy reasoning, probabilistic (logic) programming, and ASP-based integration.

Table 3
Selected frameworks retained for in-depth analysis.

Framework	GitHub repository	License	Last commit
PYLON [33]	pylon-lib/pylon	Apache 2.0	3 Dec 2021
LTN [27]/LTNtorch [34]	tommasocarraro/LTNtorch	MIT	2 Oct 2024
DomiKnowS [41]	HLR/DomiKnowS	MIT	9 Dec 2025
Scallop [16]	scallop-lang/scallop	MIT	1 May 2025
DeepProbLog [31]	ML-KULeuven/deepproblog	Apache 2.0	9 Aug 2024
SLASH [20]	ml-research/SLASH	MIT	23 May 2024
Edward [36]/Edward2 [53]	google/edward2	Apache 2.0	9 Jan 2026
DeepStochLog [39]	ML-KULeuven/deepstochlog	Apache 2.0	26 Jan 2024
NeurASP [37]	azreasoners/NeurASP	N/A	21 Oct 2023

RQ1 *Which open-source frameworks support neuro-symbolic software system development, and how can they be positioned by scope and application domain?*
The SLR retained 42 corpus articles, corresponding to 41 framework-level approaches; 19 are general-purpose, while the remaining frameworks are application-specific. Application-specific frameworks are best positioned by domain target, such as FAIR research-data management in LDM [1], biomedical entity linking in BioLinkerAI [4], or microcontroller-oriented TinyML in TinyNS [23]. General-purpose frameworks are instead positioned by reusable modelling support: languages, APIs, solvers, training interfaces, or probabilistic-programming substrates that can be transferred across various tasks. Nine of the general-purpose frameworks were selected as the most reusable subset, i.e. those with enough available documentation to support their adoption in a wide range of software development scenarios.

4. Taxonomy-Driven Qualitative Analysis

In this section, we illustrate our original taxonomy for classifying neuro-symbolic development frameworks and use it to compare the selected frameworks, listed in Table 3. The key observation behind our taxonomy is that, from the developers' viewpoint, a neuro-symbolic framework can be understood as a sequence of design commitments in their project. Developers first commit to a symbolic language, hence to a reasoning mechanism, then to a pattern of interaction between neural and symbolic components, and, finally, to a set of usability and engineering constraints. These commitments are not independent. For instance, choosing ASP brings stable-model semantics, but also grounding/solver costs. Conversely, choosing fuzzy first-order logic gives differentiable satisfaction functions, but changes the meaning of entailment. Last, using host-language constraint interfaces (e.g. Python) lowers adoption cost, but may alter the declarative specification towards a more procedural style.

The selected frameworks can be broadly categorised in four recurring families, orthogonally w.r.t. the taxonomy. Constraint-based systems such as PYLON [33] and DomiKnowS [41], fuzzy first-order systems such as LTN [27]/LTNtorch [34], (probabilistic) logic-based systems such as Scallop [16], DeepProbLog [31], SLASH [20], DeepStochLog [39], and NeurASP [37], and probabilistic-programming systems such as Edward [36]/Edward2 [53]. They all expose different combinations of language, semantics, solver, and engineering trade-offs. Table 4 summarises our analysis effort, while preserving the four taxonomy dimensions as the main interpretative key.

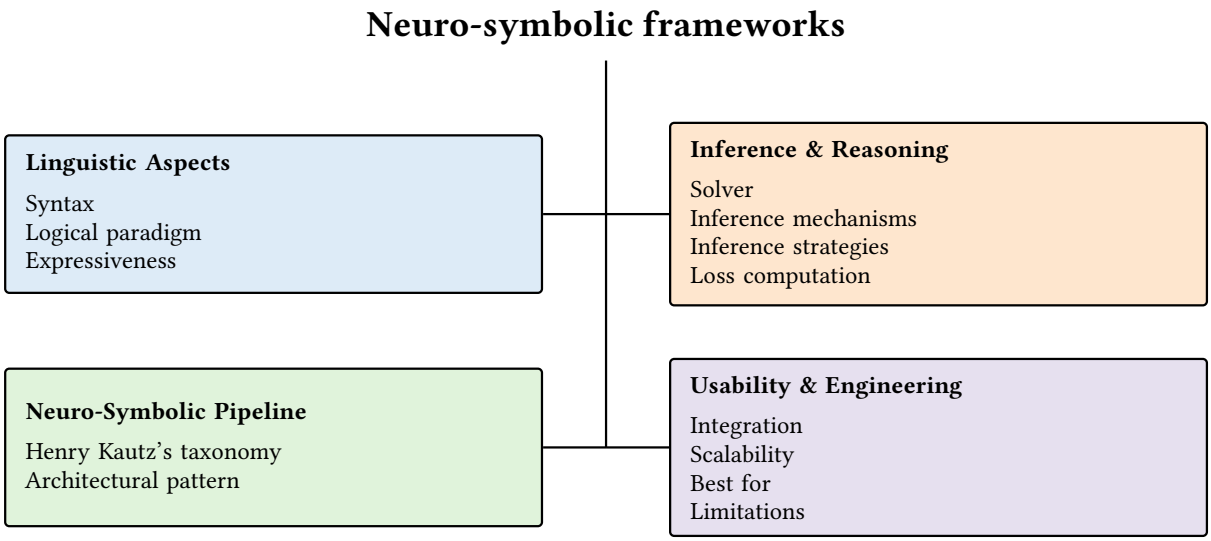

Figure 2: Taxonomy for comparing neuro-symbolic frameworks.

Figure 2 illustrates the four dimensions used to compare neuro-symbolic development frameworks. Linguistic aspects describe how symbolic knowledge is represented, namely through syntax, adopted logical paradigm, and expressiveness. Inference and reasoning capture the computational core, including the used solver, inference mechanisms and strategies, and loss computation. The neuro-symbolic pipeline dimension positions each framework with respect to Kautz's taxonomy and its architectural pattern [46]. Last, usability and engineering records practical adoption factors such as integration, scalability, best-fit use cases, and known limitations. Overall, the proposed taxonomy represents a compact map for comparing developers' design commitments rather than a checklist of isolated features. The following subsections apply the four dimensions to the selected frameworks and integrate the comparative evidence extracted from the broader review of Sect. 3.

4.1. Linguistic aspects

The first commitment in the taxonomy is linguistic. A framework must decide how symbolic knowledge is written, i.e. which logical paradigm it assumes. This dimension determines whether developers work with host-language constraints, first-order formulae, logic programmes, relational rules, stochastic computation graphs, or structured domain models. It therefore anticipates several later choices, because the language often constrains the available semantics, solver, and coding style.

Applied to the selected frameworks, the linguistic dimension separates lightweight embedding from more explicit declarative modelling. PYLON [33] exposes constraints directly inside Python, which eases adoption in neural PyTorch code but makes the logical layer less independent from procedural code. DomiKnowS [41] provides a more structured modelling interface based on constraints connected to neural sensors. LTN [27] and LTNtorch [34] use first-order formulas grounded over tensors, preserving logical vocabulary while interpreting truth through fuzzy satisfaction. These systems illustrate a first trade-off. Host-language interfaces reduce the distance from neural programming, but they make the declarative layer depend more on grounding and optimisation parameters.

Logic-based frameworks make the declarative commitment stronger. Indeed, DeepProbLog [31] extends probabilistic logic programming with neural predicates, Scallop [16] adopts a Datalog-like relational language with provenance, NeurASP [37] and SLASH [20] build on ASP-style modelling with neural or probabilistic predicates, and DeepStochLog [39] uses stochastic definite clause grammars for sequential and compositional domains. In these systems, rules define the space of derivations, stable models, proofs, or probabilistic queries over which learning and inference can operate. Last, Edward [36] and Edward2 [53] occupy a boundary position. They provide probabilistic-programming structure through random variables and probabilistic programs, but do not centre the developer experience on a logic programming language.

The resulting pattern is therefore cross-cutting. Constraint-guided systems such as PYLON [33], LTN [27], LTNtorch [34], and DomiKnowS [41] are strongest when domain knowledge is naturally expressible as consistency requirements over neural predictions. Relational and probabilistic logic systems such as DeepProbLog [31], Scallop [16], SLASH [20], DeepStochLog [39], and NeurASP [37] are strongest when the symbolic component must remain inspectable as rules, derivations, or models. Probabilistic programming in Edward [36] and Edward2 [53] is strongest when uncertainty dominates over explicit logical syntax. Overall, richer symbolic languages improve inspectability and modelling precision, but they also introduce solver-facing commitments (e.g. grounding) that also affect the reasoning and usability dimensions.

4.2. Inference & reasoning

The reasoning dimension specifies how the chosen symbolic language operates at runtime. In the taxonomy, it records the solver or computational substrate, the inference mechanism, the inference strategy, and the way reasoning contributes to the learning objective. Said otherwise, this dimension is where the linguistic commitment becomes an executable procedure.

For the selected frameworks, one major strategy is exact or compiled reasoning. DeepProbLog [31] compiles probabilistic logic programmes through sentential decision diagrams and arithmetic circuits, turning query probabilities into differentiable quantities. SLASH [20] combines stable-model reasoning with probabilistic circuits and pruning to control answer-set search. PYLON [33], in its exact mode, also relies on compilation for constraint satisfaction. DeepStochLog [39] uses AND-OR derivation forests and tabling for stochastic logic programmes, while NeurASP [37] delegates stable-model computation to CLINGO and evaluates observations through neural atoms. These approaches retain comparatively clear semantics, but their cost grows with possible worlds, derivations, constraints, or answer sets, that have to be explored.

A second strategy is relaxation into continuous optimisation. LTN [27] and LTNtorch [34] map logical connectives and quantifiers to fuzzy truth functions and work on optimising satisfaction. PYLON [33] can likewise use fuzzy or sampling-based losses. DomiKnowS [41] can express constraints as

integer linear inequalities for solver-backed prediction, but it can also support differentiable training through inference-masked or primal-dual objectives. This makes symbolic knowledge compatible with gradient-based learning, although it changes the object being computed.

Other systems make proof objects, provenance, or stochastic computation central to the reasoning interface. Scallop [16] uses provenance semirings and proof selection so that derived facts can participate in differentiable computation. Edward [36] and Edward2 [53] rely on variational inference, sampling, and gradient objectives over probabilistic programs.

The reasoning dimension is central: semantic fidelity, differentiability, and scalability are coupled, and no framework maximises all three at once. On one hand, exact probabilistic and stable-model approaches preserve more of the intended symbolic semantics, but must tame combinatorial growth of the search space. On the other hand, fuzzy approaches integrate more naturally with neural training, but trade proof-theoretic clarity for continuous satisfaction. Finally, probabilistic programming provides mature uncertainty machinery, but no logical rules by default.

4.3. Neuro-symbolic pipeline

The pipeline dimension maps the linguistic and reasoning commitments onto the architectural relation between neural and symbolic components. We use Kautz's taxonomy [46] as a vocabulary for this relation. In $\mathrm{Neuro} \mid \mathrm{Symbolic}$ systems, neural components first produce symbolic, probabilistic, or relational facts, which are then consumed by a symbolic reasoning layer. In $\mathrm{Neuro{:}Symbolic} \rightarrow \mathrm{Neuro}$ systems, symbolic knowledge mainly acts as supervision or regularisation for neural training, for instance by being compiled or relaxed into a loss. In $\mathrm{Neuro}_{\mathrm{Symbolic}}$ systems, symbolic structure is embedded tightly into differentiable computation, so that formulae, constraints, or probabilistic structure become part of the neural computation itself.

The most common pattern among the selected frameworks is $\mathrm{Neuro} \mid \mathrm{Symbolic}$. DeepProbLog [31] implements this pattern through neural predicates in probabilistic logic programmes, DeepStochLog [39] does so through stochastic grammar derivations, Scallop [16] uses differentiable relational provenance, SLASH [20] and NeurASP [37] connect neural predictions to ASP-style reasoning, and DomiKnowS [41] links neural sensors with structured domain constraints. These systems differ in language and solver, but share the same architectural commitment since a learned prediction is made available to a symbolic component for further processing.

Other frameworks instantiate tighter or differently oriented patterns. PYLON [33] is closest to $\mathrm{Neuro{:}Symbolic} \rightarrow \mathrm{Neuro}$, because symbolic constraints mainly shape the training objective through compiled or relaxed losses. LTN [27] and LTNtorch [34] are closer to $\mathrm{Neuro}_{\mathrm{Symbolic}}$, since formulae are grounded into differentiable tensor computation. Edward [36] and Edward2 [53] fit the same label only in a broad probabilistic sense. Symbolic logic is not central in those frameworks, but stochastic structure is embedded in differentiable computation.

The remaining Kautz patterns appear mainly as background cases in this corpus. To this end, application-driven frameworks make the same pipeline issues concrete. Knowledge-centric, visual, planning, event-processing, and TinyML systems discussed in Sect. 3 show that neural perception and symbolic validation are often connected through task-specific interfaces rather than through a reusable general-purpose language.

4.4. Usability & engineering

Last, but not least, the fourth dimension of the taxonomy records the engineering consequences of the previous choices. It asks how the framework is integrated, how it scales, which use cases fit its design, and which limitations matter in practice. This dimension is essential in our work as the corpus was selected around implementation-oriented frameworks, so a taxonomy of neuro-symbolic software must evaluate practical affordances as well as formal expressiveness.

Applied to the selected frameworks, usability is shaped first by the host ecosystem. PYLON [33], LTNtorch [34], Edward2 [53], and DeepStochLog [39] are comparatively easy to place inside existing

Table 4
Taxonomy view of the main general-purpose frameworks.

Framework	Language and semantics	Reasoning mechanism	Integration reading
PYLON [33]	Python constraints; Boolean, fuzzy, or probabilistic readings	Exact compilation, fuzzy relaxation, or sampling losses	$\mathrm{Neuro{:}Symbolic} \rightarrow \mathrm{Neuro}$: symbolic constraints supervise neural training.
LTN [27]/LTNtorch [34]	Fuzzy first-order logic over tensors	T-norm based satisfaction maximisation	$\mathrm{Neuro}_{\mathrm{Symbolic}}$: formulae become differentiable neural computation.
DomiKnowS [41]	Concept graphs and logical constraints	Integer constraints, inference-masked losses, primal-dual optimisation	Mainly $\mathrm{Neuro} \mid \mathrm{Symbolic}$: neural sensors are constrained by structured domain knowledge.
Scallop [16]	Datalog-like relational language with provenance	Differentiable provenance semirings and proof selection	Differentiable $\mathrm{Neuro} \mid \mathrm{Symbolic}$: derived facts and proof weights guide learning.
DeepProbLog [31]	ProbLog with neural predicates	Weighted model counting, SDDs, arithmetic circuits	$\mathrm{Neuro} \mid \mathrm{Symbolic}$: neural facts feed probabilistic logic programmes.
SLASH [20]	Neural-probabilistic ASP	Probabilistic circuits, stable-model reasoning, pruning	$\mathrm{Neuro} \mid \mathrm{Symbolic}$: neural predicates are reasoned over through ASP-style models.
Edward [36]/Edward2 [53]	Probabilistic programs over random variables	Variational inference, sampling, gradient objectives	$\mathrm{Neuro}_{\mathrm{Symbolic}}$ only in a broad probabilistic sense; logical syntax is not central.
DeepStochLog [39]	Stochastic definite clause grammars	AND-OR derivation forests and tabling	$\mathrm{Neuro} \mid \mathrm{Symbolic}$: neural predicates participate in stochastic grammar derivations.
NeurASP [37]	ASP with neural atoms	Stable-model solving and observation likelihoods	$\mathrm{Neuro} \mid \mathrm{Symbolic}$: neural outputs are probabilistic atoms in ASP.

Python machine learning workflows. LTN [27] and Edward [36] are important reference points, but their practical role is mediated by implementation choices and ecosystem evolution. DomiKnowS [41] offers a high-level modelling style, but its use of structured constraints and solver-backed inference can add configuration and debugging overhead. Source-based systems such as SLASH [20] and NeurASP [37] preserve expressive ASP-based reasoning, but might require more manual integration than frameworks distributed through standard Python packaging.

Scalability also follows the reasoning substrate. DeepProbLog [31], SLASH [20], and NeurASP [37] inherit the full expressive power of probabilistic logic or ASP, but must manage possible worlds, answer sets, grounding, or solver calls. Scallop [16] improves usability for relational reasoning through provenance and proof selection, yet still depends on controlling the size of the proof space. DeepStochLog [39] retains exact stochastic-logic inference while improving scalability through random-walk semantics, tabling, and AND-OR derivation forests.

Table 5 collects usability information extracted during our study. Particularly, it records how easily each framework can be adopted, how it scales according to its reasoning mechanism, which tasks best match its design, and which limitations are most relevant in practice. The practical lesson from our analyses of frameworks' repositories is that open-source availability is only a starting point. Documentation, packaging, maintained examples, solver dependencies, and tools for development and debugging will determine whether a framework can be reused beyond its original demonstrations.

Table 5
Usability comparison of the main general-purpose frameworks.

Framework	Accessibility	Scalability	Best suited for	Main limitations
PYLON [33]	`pip: pylon-lib`	High	Computer vision/arithmetic; structured NLP; logical games	Exact-inference cost; gradient variance.
LTN [27]/LTNtorch [34]	`pip: LTNtorch`	Medium	Semi-supervised, multi-label, and constrained-regression tasks	No probabilistic logic; grounding and quantification costs; sensitive aggregators.
DomiKnowS [41]	`pip: DomiKnowS`	Medium	Structured NLP; explainable critical systems; low-data regimes	External solvers; constraint mapping and configuration overhead.
Scallop [16]	`pip` from repository	High	Unstructured reasoning; weak supervision; VQA	Complexity grows with k; neural-output quality; recursion termination.
DeepProbLog [31]	`pip: deepproblog`	Low	Program induction; latent uncertainty; relational reasoning	Grounding, compilation, and exact-inference costs.
SLASH [20]	Source repository	High	Reasoning-based VQA; joint probabilities; set prediction	Heavy grounding; exponential solution spaces.
Edward [36]/Edward2 [53]	`pip: edward2`	High	Image transformers; large VAEs; dynamic sampling	Few high-level logical abstractions; low-level random-variable control.
DeepStochLog [39]	`pip: deepstochlog`	High	Sequences; large relational tasks; word algebra	No rule-structure learning; failed derivations.
NeurASP [37]	Source repository	Low	Semantic regularisation; commonsense reasoning; puzzles	Noisy constraints; inference bottlenecks; limited scalability.

Overall, the usability comparison shows that engineering convenience and semantic expressiveness do not always align. Standard Python packaging helps PYLON [33], LTNtorch [34], DeepProbLog [31], Edward2 [53], and DeepStochLog [39] enter existing learning pipelines, whereas source-based systems such as SLASH [20] and NeurASP [37] require more manual integration but feature well-known logic representations. Scalability is limited by stable-model inference, while pruning in SLASH [20], relaxed losses in PYLON [33] and LTN [27]/LTNtorch [34], stochastic grammars in DeepStochLog [39], and

differentiable provenance in Scallop [16] may improve scalability at the cost of some approximation.

Three cases epitomise the above considerations. LTN [27]/LTNtorch [34] is easy to adopt and differentiable by design, but universal quantification over grounded batches or concrete domains can still create a combinatorial burden, and optimisation may depend on aggregation parameters such as the p-mean. DomiKnowS [41] exposes a high-level modelling style, but solver-backed inference can introduce operational constraints, especially when commercial optimisation engines are used. Edward [36] and Edward2 [53] provide scalable probabilistic-programming, but ask developers to operate at the level of random variables rather than high-level abstractions.

The broader review of Sect. 3 also indicates two engineering issues that affect frameworks' usability indirectly. First, application-specific systems show demand for domain-facing front ends, e.g. knowledge-graph management, biomedical linking, visual reasoning, planning, event processing, TinyML, and enterprise AI all require ways to connect formal knowledge to task-specific and deployment constraints. Second, foundation-model-mediated systems suggest that natural-language interfaces can lower the cost of rule authoring, but generated specifications still require parsing and semantic validation.

RQ2 *Which linguistic, reasoning, integration, and engineering aspects distinguish available general-purpose and open-source frameworks?*
For the nine selected general-purpose open-source frameworks, the proposed taxonomy links language, reasoning semantics, pipeline, and usability. Logic-programming and ASP-based systems such as DeepProbLog [31], SLASH [20], DeepStochLog [39], and NeurASP [37] preserve proof-, derivation-, probability-, or stable-model-based semantics, but pay grounding, compilation, proof-space, or solver costs, at the price of lower scalability. Fuzzy and constraint-guided systems such as LTN [27]/LTNtorch [34], PYLON [33], and DomiKnowS [41] integrate with gradient training by replacing entailment with satisfaction, loss, or constraint violation. Edward [36]/Edward2 [53] instead provide probabilistic programming over random variables, favouring scalable uncertainty modelling over high-level logical syntax. Most selected frameworks (6 out of 9) follow a Neuro | Symbolic pipeline.

5. Open Challenges

The taxonomy-driven analysis in Sect. 4 grounds **RQ3** in a set of open challenges in the neuro-symbolic framework landscape for software development. These challenges follow from the same design commitments used in the taxonomy, i.e. the language in which knowledge is expressed, the reasoning substrate that gives that knowledge an operational meaning, the integration pattern between neural and symbolic components, and the engineering support available to developers. Most selected frameworks are best read as Neuro | Symbolic, where neural outputs feed symbolic reasoning. Consequently, the most relevant limitations cluster around three main open challenges: balancing expressiveness and scalability, improving engineering maturity and development tools, and enabling quantitative comparisons over standard benchmarks.

Expressiveness and scalability. Language expressiveness determines which kinds of symbolic knowledge a framework can represent, how directly developers can inspect that knowledge, and which reasoning costs follow from it. Logic-programming and ASP-based systems, such as DeepProbLog [31], SLASH [20], DeepStochLog [39], and NeurASP [37], preserve explicit rules, derivations, probabilistic queries, or stable models. This makes the symbolic layer semantically rich, but it also exposes the framework to the costs of searching possible worlds, proof explosion, grounding, answer-set construction, and solver calls. The same issue appears, in different forms, in relational proof construction in Scallop [16] and stochastic derivation forests in DeepStochLog [39]. The identified frameworks already encode specialised mitigations. For instance, SLASH [20] uses dynamic pruning to control answer-set search (viz., through the SAME algorithm), Scallop [16] relies on top-k proof selection, and DeepStochLog [39] combines stochastic grammars with tabling and AND-OR derivation forests. By contrast, Python-based host-language interfaces, such as PYLON [33] and LTN [27]/LTNtorch [34], lower the adoption barrier and integrate more directly with gradient-based learning, but require expressing logical knowledge

in a procedural manner. The resulting challenge is to keep knowledge specifications expressive while controlling the computational cost induced by symbolic machinery.

Engineering maturity. Our work also identifies specification effort, development tools, and engineering maturity as adoption constraints. Open-source availability is only the starting point. Indeed, release packaging, examples, documentation, development tools, solver configuration, debugging support, and integration with existing learning pipelines determine whether a framework can be reused beyond its original demonstrations in the literature. Standard Python packaging helps frameworks such as PYLON [33], LTNtorch [34], DeepProbLog [31], Edward2 [53], and DeepStochLog [39] enter existing workflows. Source-based or solver-backed systems such as SLASH [20], NeurASP [37], and DomiKnowS [41] preserve expressive reasoning interfaces, but require more effort to configure and integrate in existing workflows. Last, Prompt2DeModel [6] and VIEIRA [11] show how natural-language interfaces can reduce rule-authoring effort, although generated knowledge remains a candidate specification that must be parsed, typed, validated, and checked against the reasoning substrate.

Quantitative comparisons over standard benchmarks. Finally, the current landscape of neuro-symbolic frameworks for software development lacks an overall benchmark comparison in terms of execution times and output performance. This limits direct conclusions about how the design commitments identified by our taxonomy affect performance in comparable settings. A useful next step is therefore a controlled benchmark-based analysis, for instance on MNIST-based tasks [57], to assess execution times, result accuracy and developers' effort across different frameworks.

RQ3 *Which limitations and open challenges characterise the current framework landscape?*
The nine-framework comparison highlights three main challenges, i.e. balancing language expressiveness with scalability, improving engineering maturity and development tools, and performing controlled quantitative comparisons among frameworks. Logic- and ASP-based frameworks such as DeepProbLog [31], SLASH [20], DeepStochLog [39], and NeurASP [37] preserve explicit symbolic semantics, but their scalability may be limited by possible worlds, proof explosion, grounding, answer-set construction, and solver calls. Some mitigations already exist, including dynamic pruning in SLASH [20], top-k proof selection in Scallop [16], and stochastic grammars with tabling and AND-OR derivation forests in DeepStochLog [39]. Conversely, Python-based interfaces such as PYLON [33] and LTN [27]/LTNtorch [34] ease adoption but move logical knowledge closer to procedural implementation. Engineering maturity remains uneven, since reuse depends on packaging, documentation, examples, solver configuration, debugging support, and validation of generated declarative specifications such as those produced by Prompt2DeModel [6]. Future work should focus on improving the scalability of expressive solutions, releasing adequate development support tools (i.e. IDE, debuggers), and comparing existing frameworks on standard benchmarks.

6. Concluding remarks

This article has presented an SLR and taxonomy-driven analysis of open-source frameworks for developing neuro-symbolic software systems. We first collected 42 corpus articles, corresponding to 41 framework-level approaches. We distinguished application-specific frameworks from general-purpose development frameworks, and analysed in depth nine general-purpose open-source frameworks with suitable documentation support. The proposed taxonomy provides a compact comparative lens for relating linguistic choices, reasoning mechanisms, integration patterns, and engineering constraints, making framework trade-offs explicit from a developer's viewpoint. We finally identified three main open challenges: balancing expressiveness and scalability, improving engineering maturity and development tools, and performing quantitative comparisons of the available frameworks over standard benchmarks. Future work will address the latter through controlled experiments on a standard benchmark, such as MNIST-based tasks [57], to compare the selected frameworks under quantitative criteria.

Declaration on Generative AI

During the preparation of this work, the Authors used GPT-5.5 to perform the following activities: grammar and spelling check, paraphrase and reword, and improve writing style. The Authors reviewed and edited all content as needed and take full responsibility for the publication's content.

References

[1] Sakor, Ahmad, Brunet, Mauricio, Iglesias, Enrique, Rivas, Ariam, Rohde, Philipp D, Kraft, Angelina, Vidal, Maria-Esther. Integrating Knowledge Graphs and Neuro-Symbolic AI: LDM Enables FAIR and Federated Research Data Management. *Proceedings of the Eighteenth ACM International Conference on Web Search and Data Mining*, 1044–1047, 2025.

[2] Manigrasso, Francesco, Lamberti, Fabrizio, Morra, Lia. Boosting zero-shot learning through neuro-symbolic integration. *Pattern Recognition*, 170, 111869, 2026.

[3] Kamali, Danial, Barezi, Elham J, Kordjamshidi, Parisa. Nesycoco: A neuro-symbolic concept composer for compositional generalization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39, 4184–4193, 2025.

[4] Sakor, Ahmad, Singh, Kuldeep, Vidal, Maria-Esther. BioLinkerAI: Leveraging LLMs to Improve Biomedical Entity Linking and Knowledge Capture. *Proceedings of the Eighteenth ACM International Conference on Web Search and Data Mining*, 1110–1111, 2025.

[5] Bueff, Andreas, Belle, Vaishak. Learning explanatory logical rules in non-linear domains: a neuro-symbolic approach. *Machine Learning*, 113(7), 4579–4614, 2024.

[6] Faghihi, Hossein Rajaby, Nafar, Aliakbar, Uszok, Andrzej, Karimian, Hamid, Kordjamshidi, Parisa. Prompt2DeModel: Declarative Neuro-Symbolic Modeling with Natural Language. *International Conference on Neural-Symbolic Learning and Reasoning*, 315–327, 2024.

[7] Smirnov, Alexander, Ponomarev, Andrew, Agafonov, Anton. Ontology-Based Neuro-Symbolic AI: Effects on Prediction Quality and Explainability. *IEEE Access*, 2024.

[8] Bober-Irizar, Mikel, Banerjee, Soumya. Neural networks for abstraction and reasoning. *Scientific Reports*, 14(1), 27823, 2024.

[9] Cosler, Matthias, Hahn, Christopher, Omar, Ayham, Schmitt, Frederik. Neurosynt: A neuro-symbolic portfolio solver for reactive synthesis. *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 45–67, 2024.

[10] Glauer, Martin, Neuhaus, Fabian, Flügel, Simon, Wosny, Marie, Mossakowski, Till, Memariani, Adel, Schwerdt, Johannes, Hastings, Janna. Chebifier: automating semantic classification in ChEBI to accelerate data-driven discovery. *Digital Discovery*, 3(5), 896–907, 2024.

[11] Li, Ziyang, Huang, Jiani, Liu, Jason, Zhu, Felix, Zhao, Eric, Dodds, William, Velingker, Neelay, Alur, Rajeev, Naik, Mayur. Relational programming with foundational models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38, 10635–10644, 2024.

[12] Giunchiglia, Eleonora, Tatomir, Alex, Stoian, Mihaela Cătălina, Lukasiewicz, Thomas. CCN+: A neuro-symbolic framework for deep learning with requirements. *International Journal of Approximate Reasoning*, 171, 109124, 2024.

[13] van Krieken, Emile, Badreddine, Samy, Manhaeve, Robin, Giunchiglia, Eleonora. ULLER: A Unified Language for Learning and Reasoning. *International Conference on Neural-Symbolic Learning and Reasoning*, 2024.

[14] Aspis, Yaniv, Albinhassan, Mohammad, Lobo, Jorge, Russo, Alessandra. Embed2rule scalable neuro-symbolic learning via latent space weak-labelling. *International Conference on Neural-Symbolic Learning and Reasoning*, 195–218, 2024.

[15] Manigrasso, Francesco, Schouten, Stefan, Morra, Lia, Bloem, Peter. Probing llms for logical reasoning. *International Conference on Neural-Symbolic Learning and Reasoning*, 257–278, 2024.

[16] Li Z.; Huang J.; Naik M.. Scallop: A Language for Neurosymbolic Programming. *Proceedings of the ACM on Programming Languages*, 2023.

[17] Cunnington, Daniel, Law, Mark, Lobo, Jorge, Russo, Alessandra. Neuro-symbolic learning of answer set programs from raw data. *IJCAI International Joint Conference on Artificial Intelligence*, 2023.

[18] Aditya, Dyuman, Mukherji, Kaustuv, Balasubramanian, Srikar, Chaudhary, Abhiraj, Shakarian, Paulo. PyReason: Software for open world temporal logic. *CEUR Workshop Proceedings*, 2023.

[19] Manigrasso, Francesco, Miro, Filomeno Davide, Morra, Lia, Lamberti, Fabrizio. Faster-LTN: a neuro-symbolic, end-to-end object detection architecture. *International Conference on Artificial Neural Networks*, 40–52, 2021.

[20] Skryagin A.; Ochs D.; Dhami D.S.; Kersting K.. Scalable Neural-Probabilistic Answer Set Programming. *Journal of Artificial Intelligence Research*, 2023.
[21] A. P. Sheth; K. Roy; R. Venkataramanan; V. Nadimuthu; C. Shyalika. Composite AI With Custom, Compact, Neurosymbolic Models: The Emergent Enterprise Artificial Intelligence Paradigm. *IEEE Internet Computing*, 2025.
[22] Acharya, Kamal, Lad, Mehul, Sun, Liang, Song, Houbing. Neurosymbolic Approach for Travel Demand Prediction: Integrating Decision Tree Rules into Neural Networks. *2025 International Wireless Communications and Mobile Computing (IWCMC)*, 600–605, 2025.
[23] Saha, Swapnil Sayan, Sandha, Sandeep Singh, Aggarwal, Mohit, Wang, Brian, Han, Liying, Briseno, Julian De Gortari, Srivastava, Mani. TinyNS: Platform-aware neurosymbolic auto tiny machine learning. *ACM Transactions on Embedded Computing Systems*, 23(3), 1–48, 2024.
[24] Zhang, Yichi, Yang, Jianing, Pan, Jiayi, Storks, Shane, Devraj, Nikhil, Ma, Ziqiao, Yu, Keunwoo, Bao, Yuwei, Chai, Joyce. Danli: Deliberative agent for following natural language instructions. *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 1280–1298, 2022.
[25] Khan, M Jaleed, Breslin, John G, Curry, Edward. KnowZRel: Common Sense Knowledge-based Zero-Shot Relationship Retrieval for Generalised Scene Graph Generation. *IEEE Transactions on Artificial Intelligence*, 2025.
[26] Smirnova, Alisa, Yang, Jie, Yang, Dingqi, Cudre-Mauroux, Philippe. Nessy: A neuro-symbolic system for label noise reduction. *IEEE Transactions on Knowledge and Data Engineering*, 35(8), 8300–8311, 2022.
[27] Badreddine S.; d'Avila Garcez A.; Serafini L.; Spranger M.. Logic Tensor Networks. *Artificial Intelligence*, 2022.
[28] Sheth, Amit, Khandelwal, Vishal, Roy, Kaushik, Pallagani, Vishal, Chakraborty, Moumita. NeuroSymbolic Knowledge-Grounded Planning and Reasoning in Artificial Intelligence Systems. *IEEE Intelligent Systems*, 2025.
[29] Shindo, Hikaru, Pfanschilling, Viktor, Dhami, Devendra Singh, Kersting, Kristian. α ilp: thinking visual scenes as differentiable logic programs. *Machine Learning*, 112(5), 1465–1497, 2023.
[30] Skryagin, Arseny, Stammer, Wolfgang, Ochs, Daniel, Dhami, Devendra Singh, Kersting, Kristian. Neural-Probabilistic Answer Set Programming. *19th International Conference on Principles of Knowledge Representation and Reasoning, KR 2022*, 2022.
[31] Manhaeve R.; Dumancic S.; Kimmig A.; Demeester T.; De Raedt L.. Neural probabilistic logic programming in DeepProbLog. *Artificial Intelligence*, 2021.
[32] Vilamala, Marc Roig, Xing, Tianwei, Taylor, Harrison, Garcia, Luis, Srivastava, Mani, Kaplan, Lance, Preece, Alun, Kimmig, Angelika, Cerutti, Federico. Using deepproblog to perform complex event processing on an audio stream. *Proceedings of the 10th International Workshop on Statistical Relational AI*, 2021.
[33] Ahmed K.; Li T.; Ton T.; Guo Q.; Chang K.-W.; Kordjamshidi P.; Srikumar V.; Van den Broeck G.; Singh S.. PYLON: A PyTorch Framework for Learning with Constraints. *Proceedings of the 36th AAAI Conference on Artificial Intelligence, AAAI 2022*, 2022.
[34] Tommaso Carraro. LTNtorch: PyTorch implementation of Logic Tensor Networks. *LTNtorch: PyTorch implementation of Logic Tensor Networks*, 2025.
[35] Martone, Simone, Manigrasso, Francesco, Lamberti, Fabrizio, Morra, Lia. Prototypical logic tensor networks (proto-ltn) for zero shot learning. *2022 26th International Conference on Pattern Recognition (ICPR)*, 4427–4433, 2022.
[36] Dustin Tran, Matthew D. Hoffman, Rif A. Saurous, Eugene Brevdo, Kevin Murphy, David M. Blei. Deep probabilistic programming. *International Conference on Learning Representations*, 2017.
[37] Yang Z.; Ishay A.; Lee J.. NeurASP: Embracing neural networks into answer set programming. *IJCAI International Joint Conference on Artificial Intelligence*, 2020.
[38] Eiter, Thomas, Higuera, Nelson, Oetsch, Johannes, Pritz, Michael. A neuro-symbolic ASP pipeline for visual question answering. *Theory and Practice of Logic Programming*, 22(5), 739–754, 2022.
[39] Winters T.; Marra G.; Manhaeve R.; De Raedt L.. DeepStochLog: Neural Stochastic Logic Programming. *Proceedings of the 36th AAAI Conference on Artificial Intelligence, AAAI 2022*, 2022.
[40] Pryor, Connor, Dickens, Charles, Augustine, Eriq, Albalak, Alon, Wang, William, Getoor, Lise. Neupsl: Neural probabilistic soft logic. *IJCAI International Joint Conference on Artificial Intelligence*, 2022.
[41] Rajaby Faghihi, Hossein, Guo, Quan, Uszok, Andrzej, Nafar, Aliakbar, Kordjamshidi, Parisa. DomiKnowS: A Library for Integration of Symbolic Domain Knowledge in Deep Learning. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 231–241, 2021.
[42] Cunnington, Daniel, Law, Mark, Lobo, Jorge, Russo, Alessandra. FFNSL: feed-forward neural-symbolic learner. *Machine Learning*, 112(2), 515–569, 2023.
[43] Colelough, Brandon C, Regli, William. Neuro-symbolic AI in 2024: A systematic review. *arXiv preprint*

arXiv:2501.05435, 2025.

[44] Garcez, Artur d'Avila, Lamb, Luis C. Neurosymbolic ai: The 3rd wave. *Artificial Intelligence Review*, 56(11), 12387–12406, 2023.

[45] Hamilton, Kyle, Nayak, Aparna, Božić, Bojan, Longo, Luca. Is neuro-symbolic AI meeting its promises in natural language processing? A structured review. *Semantic Web*, 15(4), 1265–1306, 2024.

[46] Kautz, Henry. The third ai summer: Aaai robert s. engelmore memorial lecture. *Ai magazine*, 43(1), 105–125, 2022.

[47] Krippendorff, Klaus. Content Analysis: An Introduction to Its Methodology 2 ed., *Sage*, 2004.

[48] Lu, Zhen, Afridi, Imran, Kang, Hong Jin, Ruchkin, Ivan, Zheng, Xi. Surveying neuro-symbolic approaches for reliable artificial intelligence of things. *Journal of Reliable Intelligent Environments*, 10(3), 257–279, 2024.

[49] Marra, Giuseppe, Dumančić, Sebastijan, Manhaeve, Robin, De Raedt, Luc. From statistical relational to neurosymbolic artificial intelligence: A survey. *Artificial Intelligence*, 328, 104062, 2024.

[50] Nawaz, Uzma, Anees-ur-Rahaman, Mufti, Saeed, Zubair. A review of neuro-symbolic AI integrating reasoning and learning for advanced cognitive systems. *Intelligent Systems with Applications*, 26, 200541, 2025.

[51] Petersen, Kai, Feldt, Robert, Mujtaba, Shahid, Mattsson, Michael. Systematic mapping studies in software engineering. *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE'08)*, 68–77, 2008.

[52] Petersen, Kai, Vakkalanka, Sairam, Kuzniarz, Ludwik. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64, 1–18, 2015.

[53] Tran, Dustin, Hoffman, Matthew D., Moore, Dave, Suter, Christopher, Vasudevan, Srinivas, Radul, Alexey, Johnson, Matthew, Saurous, Rif A. Simple, Distributed, and Accelerated Probabilistic Programming. *Advances in Neural Information Processing Systems*, 2018.

[54] Valiant, Leslie G. Three problems in computer science. *Journal of the ACM (JACM)*, 50(1), 96–99, 2003.

[55] Wan, Zishen, Liu, Che-Kai, Yang, Hanchen, Li, Chaojian, You, Haoran, Fu, Yonggan, Wan, Cheng, Krishna, Tushar, Lin, Yingyan, Raychowdhury, Arijit. Towards cognitive ai systems: a survey and prospective on neuro-symbolic ai. *arXiv preprint arXiv:2401.01040*, 2024.

[56] Wohlin, Claes and Runeson, Per and Höst, Martin and Ohlsson, Magnus C and Regnell, Björn and Wesslén, Anders and others. Experimentation in software engineering *Springer*, 236, 2012.

[57] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]," *IEEE Signal Processing Magazine*, 29(6), 141–142, 2012.