A Survey of Neurosymbolic Answer Set Programming

Journal Title
XX(X):1–48
©The Author(s) 2025
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Alexander Philipp Rader[1] and Alessandra Russo[1]

Abstract
Neurosymbolic artificial intelligence (AI) combines neural networks and symbolic methods to create robust and explainable frameworks. This survey provides an overview of the literature on neurosymbolic AI that uses answer set programming (ASP) as its symbolic language of choice. ASP is a logical formalism that can represent expressive rules and common-sense reasoning in a compact and human-readable form. Bridging the gap between neural representations and categorical symbols is a difficult task, especially when the learning of knowledge is involved. Many approaches have been proposed in the field to overcome these challenges and we categorise them based on which components are hard-coded or learned. We provide illustrations and explanations of the different types of frameworks and compare them with each other. We discuss the advantages of such hybrid models in terms of explainability and logical robustness. Lastly, we explore the limits of the field, including the simplicity of tasks, the extensive use of hard-coded knowledge, and the limited scalability of methods. We argue that better benchmarks, improvements in scalability and novel ways of propagating the learning signal through ASP components are needed to propel the field forward.

Keywords
Answer Set Programming, Neurosymbolic AI

Introduction

As AI systems are deployed more widely, issues of trust, safety and interpretability become ever more important. Modern neural AI models have made strides in many areas such as holding conversations (Liu et al. 2023b), passing difficult exams (OpenAI 2023) and generating images from text prompts (Podell et al. 2024). They are remarkably capable of processing real-world data and autonomously learning

[1]Department of Computing, Imperial College London, United Kingdom

Corresponding author:
Alexander Philipp Rader, Imperial College London, Exhibition Rd, London SW7 2AZ, UK.
Email: apr20@ic.ac.uk

knowledge from examples. However, they lack explicit reasoning and logic capabilities, which can lead to inconsistent outputs and hallucinations (Farquhar et al. 2024). Their black-box nature also means they are not explainable and lack formal guarantees (John-Mathews 2021).

Marcus (2020) argues that symbolic representations are necessary for AI to achieve robust reasoning. Unlike neural networks, symbolic AI acquires an internal model to represent abstract knowledge and can reason with it logically. This model is human-readable, making its decisions transparent and explainable. Depending on the formal representation of the model, guarantees can be made about its behaviour as well. However, symbolic methods struggle to deal with real-world, noisy data and often scale exponentially with the size of the problem domain.

The intersection of neural networks and symbolic methods is known as neurosymbolic AI and aims to combine the best of both worlds (Garcez and Lamb 2023). There are countless ways of integrating these two paradigms. In this survey, we focus on the use of answer set programming (ASP), a type of symbolic AI that belongs to the field of logic programming. ASP is more expressive than languages like Prolog, while still being relatively efficient to compute (Lifschitz 2019). It is therefore well suited for representing complex tasks and a popular choice for neurosymbolic AI. We explore the capabilities, advantages and drawbacks of frameworks combining ASP and neural networks throughout this survey.

We start by positioning this survey among other overview papers for similar fields in the related work section. Additionally, we outline the use of other symbolic languages in neurosymbolic AI and compare them with ASP, with a particular focus on formalisms within inductive logic programming. We establish that this survey focuses on frameworks combining neural networks with ASP that were published within the last six years. The remainder of the related work section includes pointers to work combining ASP with reinforcement learning and application papers. In the background section, we formally define ASP, explain how to learn ASP rules and provide a quick overview of neural networks.

The *Neurosymbolic ASP frameworks* section contains a discussion of papers in the field of neurosymbolic ASP. It is split into four parts, which represent different categories of frameworks. We categorise frameworks based on whether their neural component is pre-trained, their symbolic component is hard-coded, or either of them is learned. We briefly describe each framework and discuss its strengths and weaknesses. The descriptions are accompanied by illustrations allowing the reader to compare and contrast them.

In the *Analysis of neurosymbolic ASP* section, we evaluate the performance of papers and identify the current strengths, limits and open challenges in the field. The section is split into four parts: assessing the difficulty of perception tasks, comparing accuracies on datasets, identifying the limits of ASP generation and analysing scalability issues. Overall, the neurosymbolic frameworks perform well, often beating fully neural baselines in terms of accuracy in addition to explainability and robustness. However, most datasets contain simple inputs that do not reflect real-world scenarios. Another weakness is the reliance on hand-written background knowledge and the need to heavily restrict the search space for finding ASP rules. Lastly, the field suffers from timeouts and scalability issues that limit the ability of frameworks to scale to real-world tasks.

We conclude that a great deal of research effort has been devoted to neurosymbolic ASP in recent years. To continue pushing the field forward, more challenging datasets and novel methods for learning are needed. A promising new direction is the combination of foundation models and ASP, where the two paradigms complement each other well. ASP can help models move beyond summarising existing knowledge towards making new discoveries while being robust and explainable.

Related work

In this section, we position our work among existing survey papers and related fields. We discuss the focus of our survey on ASP and examine its strengths and weaknesses compared to other logic programming paradigms. We also review related literature and provide pointers to adjacent work.

Neurosymbolic AI

This survey is situated within the wider field of neurosymbolic AI. Belle and Marcus (2026) provide a historical perspective on the subject, highlight key developments, and conclude with an outlook into the future. High-level overviews of approaches can be found in Hitzler et al. (2022) and Sheth et al. (2023), which serve as a broad introduction.

For more comprehensive reviews, there are a variety of surveys that cover the field in detail: Colelough and Regli (2025) conduct a systematic search of all papers published about neurosymbolic AI between 1970 and 2024. They uncover a large increase in publications on this topic beginning in 2020. Gibaut et al. (2023) focus on six methods in neurosymbolic AI and survey the associated papers. Yu et al. (2023) give a comprehensive overview of the domain, including its taxonomy, techniques and applications. Wan et al. (2024a) group systems into five categories and profile them based on metrics such as underlying operations. Hitzler and Sarker (2021) include a collection of 17 overview papers, encompassing different branches of the field. The methods include graph reasoning, boolean circuits or logic tensor networks. Last but not least, Bhuyan et al. (2024) create a conceptual map of the papers in the field, categorising them based on domain, type and properties. They also delve deeper into individual frameworks, breaking down the kinds of reasoning, representations and logics they use.

Many survey papers focus on specific aspects of the field. Marra et al. (2024) explore neurosymbolic AI together with statistical relational AI. They identify shared dimensions between them and position frameworks along these dimensions. Acharya and Song (2025) look at the field through the lens of robustness, uncertainty quantification and intervenability. They classify how different techniques enable improvements in these areas and outline current challenges. Wan et al. (2024b) experimentally evaluate a suite of neurosymbolic algorithms in terms of metrics such as runtime and memory usage. They uncover bottlenecks caused by factors such as symbolic operations, data dependencies, or complex flow control. Bouneffouf and Aggarwal (2022) focus on applications of neurosymbolic AI in fields including healthcare, finance, and information retrieval. Manhaeve et al. (2026) provide an overview of popular benchmarks used to evaluate frameworks. They study the strengths and limitations of these tasks used and propose desirable features that new benchmarks should have. Lamb et al. (2021) explore the use of graph neural networks in neurosymbolic domains, including developments and applications. Odense and d'Avila Garcez (2025) take a first step towards a formalisation of neurosymbolic AI. They establish a formal definition of semantic encodings and show that many methods already fall under it. Smet and Raedt (2025) contribute definitions for neurosymbolic models and inference. They build on the observation that such systems combine logic with beliefs and show that many classes of models can be cast within their definitions. These recent papers demonstrate growing efforts to establish a unifying formalisation of the field and its many approaches.

We discussed surveys that provide a comprehensive overview of different methods in neurosymbolic AI. Due to their broad scope, they tend to focus on major streams of research. There is no survey that details the structures, strengths and limitations of frameworks that use ASP. The closest contender is the work by Borroto et al. (2025), which reports on the intersection between ASP and neurosymbolic AI. However, the authors limit their focus on a selection of frameworks and applications, leaving room for a more comprehensive survey. Our work aims to fill this gap and provide a detailed account of the state of the art in neurosymbolic ASP.

Symbolic languages

The *symbolic* part of neurosymbolic AI encompasses a wide range of different languages and paradigms. In this section, we highlight some of the most common symbolic methods and discuss how they have been combined with neural networks. For a more thorough overview, we direct you to the surveys discussed in the previous section. Finally, we position ASP within the paradigm of logic programming and contrast it with other languages in that category.

Propositional and first-order logic. A fundamental branch of symbolic logic is propositional logic, which deals with two-valued variables combined using conjunctions, disjunctions and negation. There are many frameworks encoding propositional formulas within layers of neural networks to make them more explainable, such as Pix2Rule (Cingillioglu and Russo 2021; Baugh et al. 2023), logic gate networks (Petersen et al. 2022) or TELL (Ragno et al. 2024). First-order logic increases expressivity by adding quantifiers, predicates and variables for representing complex knowledge. There are many efforts to integrate first-order logic with neural networks, including logic explained networks (Ciravegna et al. 2023), logical neural networks (Riegel et al. 2020) and neural logic machines (Dong et al. 2019). Badreddine et al. (2022) propose logic tensor networks and define their own language called *real logic* for it. It is a variant of first-order logic with fuzzy semantics, where domains are represented by tensors and reasoning is implemented with real-valued tensor operations.

Symbolic sentences can take different forms that are useful for modelling and solving problems. In decision trees, logical propositions are found in nodes and their truth values determine which edges to take up to a leaf node that represents a classification. Frameworks that propose neurosymbolic decision trees are Möller et al. (2025) and Kairgeldin and Carreira-Perpiñán (2025). Probabilistic circuits can encode logic formulas into computational graphs that represent probability distributions and support exact inference (Choi et al. 2020). They have been integrated into neural networks as a probabilistic layer that ensures that predictions satisfy logical constraints (Ahmed et al. 2022). Other logical formalisms with neurosymbolic frameworks include deterministic finite automata (Zhang et al. 2024), planning domain definition language (Liu et al. 2023a) and action language (Ishay and Lee 2025).

Logic programming. A widely-used symbolic paradigm is logic programming, which models computations as inference through a program. Such a program consists of facts and rules, in which the head holds true whenever the body holds true. A query can be solved by checking whether it holds true given the program. ASP belongs to the family of logic programming, along with languages like Prolog, Datalog and abductive logic. There has been great interest in the research community to combine logic programming with neural networks.

Prolog is the quintessential logic programming language. It supports representing knowledge as definite clauses (with one head literal) and solves queries in a top-down, goal-directed fashion (Clocksin and Mellish 1981). ProbLog is an extension to Prolog that allows facts to be annotated with probabilities and therefore model uncertainty (De Raedt et al. 2007). By specifying these probabilities dynamically using neural networks, Manhaeve et al. (2021) create the neurosymbolic framework DeepProbLog.

Given a program, DeepProbLog enables end-to-end training of the neural network by calculating the gradients of probabilistic facts and backpropagating them to the network. The result is a unified framework that integrates symbolic knowledge with the capabilities of neural networks to classify real-world data. There are numerous variants and extensions of DeepProbLog that overcome shortcomings or enhance the capabilities of the framework: DeepSeaProbLog adds continuous probability distributions by incorporating weighted model integration (De Smet et al. 2023). DeepStochLog integrates neural networks into stochastic definite clause grammars, which scales better than DeepProbLog as it does not have to enumerate all possible worlds (Winters et al. 2022). VAEL combines ProbLog with variational autoencoders, introducing symbolic and neural atoms to the latent space to aid generating the desired output (Misino et al. 2022).

Datalog is a function-free subset of Prolog originally built for handling databases. Its restrictions guarantee that queries will terminate and allow it to apply efficient bottom-up evaluation strategies for better scalability. Scallop uses a probabilistic database as the interface between the neural and symbolic components. The neural network predictions are stored as facts in the database and then queried over to find the downstream label. By restricting logical evaluation to the top-k proofs, Scallop can calculate gradients for the neural component efficiently (Huang et al. 2021). The apperception engine uses a temporal variation of Datalog, which incorporates causal rules to model transitions. The framework builds a theory that predicts future states given temporal data (Evans et al. 2021). Other variations include neural Markov Prolog, which compiles logic rules into a Markov network (Thomson and Page 2023), and CR-Prolog, a non-monotonic extension of Prolog that has been used for visual question answering (Riley and Sridharan 2019).

Logical abduction extends standard logic programming by inferring facts and hypotheses that best explain an observation. The Abductive Learning (ABL) framework implements abduction to bridge the gap between neural predictions and a logical program. Given background knowledge and pseudo-labels from the neural network, the abductive procedure evaluates whether the neural predictions are consistent with the observation and provides corrected labels if not (Dai et al. 2019). MetaABD is an extension to ABL that trains a neural network and induces the logical theory jointly. The background knowledge is not given in full, instead the framework uses a combination of abduction and induction to learn recursive first-order theories with predicate invention (Dai and Muggleton 2021). Han et al. (2023) employ abductive logical reasoning to induce hypotheses for unpartitioned data. These hypothesis capture the relationship between the target and subconcepts, which in turn are used to train neural networks. Similar to VAEL, MetaAbd has been applied to increase control over visual generation with logical symbols (Peng et al. 2025). The method has been improved with pre-training to reduce the search space and achieve speed-ups (Jin et al. 2025).

ASP introduces constructs into logic programming that help model commonsense reasoning with incomplete knowledge. It supports default negation, where an atom is assumed false unless there is evidence for it, and non-monotonic reasoning, where additional evidence can retract existing conclusions. For these reasons, it is more expressive than languages like Prolog, where rules are limited to definite logic, or MetaAbd, which cannot learn hard constraints. Law et al. (2018) prove that the learning of answer set programs subsumes the other paradigms in inductive logic programming, making learning in ASP the most general inductive framework.

However, expressivity is not the only relevant factor for neurosymbolic frameworks. More restricted languages tend to run faster and can be easier to use. For example, Datalog can use efficient database algorithms to scale better than Prolog. Abductive methods can conduct a guided search over the logic program space to find latent labels, rather than enumerating all world possibilities. The query-driven nature of Prolog can make framework design more intuitive than the stable model semantics of ASP. Different frameworks also contain unique features, such as modelling probabilistic facts in ProbLog or continuous probability distributions in DeepSeaProbLog. Many tasks do not make use of the higher expressivity that ASP offers, often containing simple programs such as addition. In such cases, it is desirable to choose frameworks with sufficient expressivity and superior scalability. There is no single best language for neurosymbolic frameworks, as they all involve trade-offs.

We have discussed a wide range of symbolic languages for neurosymbolic AI in this section. To enable us to cover each framework in sufficient detail and allow comparative analyses, this survey will focus on ASP as the logical language. ASP is a popular choice due to its ability to model commonsense reasoning while still being computable by efficient solvers. The many-worlds semantics of ASP brings with it unique opportunities and challenges, making it a worthwhile subject of study for this survey. To keep the discussions relevant, we restrict ourselves to papers released within the last six years, i.e. from 2020 onwards. Moreover, we restrict our focus to "classical" learning of neural networks with supervised or semi-supervised methods. For prominent work on combining reinforcement learning and ASP, we provide a brief overview in the next section.

Reinforcement learning

ASP is also a popular specification language for rewards in reinforcement learning (RL). A variety of papers define goals in ASP and some even learn rules from traces.

Agostinelli et al. (2024) specify the set of goal states with ASP, rather than listing each state one by one. A deep reinforcement learning algorithm then estimates a heuristic function of the distance from the current state to this set of goal. Albilani and Bouzeghoub (2023) use ASP rules in two ways: to generate traces for learning low-level policies and as a backup policy in safety-critical scenarios. Tudor and Gupta (2024) specify rules in the goal-directed ASP variant s(CASP) for RL and use a dependency graph to prune the rules for faster execution. In this way, ASP helps train the neural component.

Leonetti et al. (2016) use ASP to represent the transition model of an environment and calculate plans using an ASP solver. The plans represent partial policies, which restrict an agent to reasonable actions during execution. Among these actions, the agent learns the expected cumulative reward using RL. In principle, any planner can be used, but the authors chose ASP due to its ability to represent defaults. It allows them to compactly represent optimistic assumptions, e.g. that all doors are open unless proven otherwise.

Furelos-Blanco et al. (2021) introduce ISA, a framework for learning automata for subgoals in RL. The automata are presented in ASP and have states for achieving or failing high-level goals. They use ILASP to learn these automata from RL traces. Parać et al. (2024) extend this work and introduce the ability to handle noisy data.

Applications

Neurosymbolic ASP has been applied in a variety of settings and we highlight some of them in this section. For a more thorough review of applications in neurosymbolic AI more generally, refer to Bouneffouf and Aggarwal (2022).

Suchan et al. (2021) combine object detection with spatial-temporal prediction and ASP for visual sensemaking in autonomous driving. The environment is modelled with relational representations pertaining to space and motion, while ASP computes commonsense interpretations about safety, space and change. As it is applied to self-driving, the framework works online in a perceive-interpret-decide cycle. Rajasekharan et al. (2023a) employ LLMs and ASP for argumentation analysis. They extract the argumentation structure of a set of documents and represent it as an answer set program, which is used to prove a claim. As an application, they demonstrate their framework on the topic of the MH17 Malaysian Airlines flight downing. Barbara et al. (2023) propose an industrial application for neurosymbolic ASP in the compliance checking of electrical control panels. They use deep learning to recognize electrical components in images of panels and reconstructing its scheme. ASP is then employed to compare the reconstructed scheme with the original schematic to discover problems. The authors prove the systems by deploying it on a real test case in electrical control panel construction. Lastly, Chu-Carroll et al. (2024) have used neuro-symbolic ASP in real-world applications at their company Elemental AI. They use it for solving constraint satisfaction and optimisation problems. ASP serves as the logical reasoning engine and LLMs are used for knowledge acquisition and user interaction, in what they call an "LLM sandwich". For knowledge acquisition, the LLM translates user inputs into an intermediate language, called Cogent, which is a constrained subset of English. For user interaction, the LLM takes the output of the ASP reasoner and presents it in a natural language interface.

Background

Answer set programming (ASP)

We give a brief overview of the syntax and semantics of ASP. For detailed definitions, please refer to Gelfond and Kahl (2014) and Lifschitz (2019).

Syntax. The language of ASP consists of constants, functions, predicates and variables. A **term** can be constructed as follows:

- A variable or constant is a term,
- If $\mathtt{t_1}, \ldots, \mathtt{t_n}$ are terms and $\mathtt{f}$ is a function of arity n, then $\mathtt{f}(\mathtt{t_1}, \ldots, \mathtt{t_n})$ is a term.

An **atom** is an expression of the form $\mathtt{p}(\mathtt{t_1}, \ldots, \mathtt{t_n})$ where $\mathtt{p}$ is a predicate of arity n. If $n = 0$, we omit the parentheses and just write $\mathtt{p}$. Terms without variables are called **ground** and an atom is ground if every term in it is ground. A set of ground atoms is called an interpretation.

A general **rule** consists of atoms $\mathtt{h_1}, \ldots, \mathtt{h_k}$ and $\mathtt{b_1}, \ldots, \mathtt{b_n}$ and has the form:

$$\mathtt{h_1} \vee \cdots \vee \mathtt{h_k} \mathrel{:-} \mathtt{b_1}, \ldots, \mathtt{b_m}, \mathtt{not}\ \mathtt{b_{m+1}}, \ldots \mathtt{not}\ \mathtt{b_n}$$

The left part of the rule is the **head** and the right part is the **body**. The head is a disjunction of atoms and the symbol $\vee$ is read as *or*. If the head only contains one atom, it is a **normal** rule. If the head is empty, the rule is a **constraint**. The body contains positive and negative atoms, the latter are indicated by the symbol `not` in front of them. Unlike classical negation, the symbol `not` denotes **negation as failure** and means that an atom is not believed to be true. If there are no negative atoms in the body, it is a **definite** rule. A **program** is a collection of rules. A definite program consists of only definite rules. A tight program contains no positive cycles and a stratified program contains no cycles through negation.

Semantics. A set S of ground atoms satisfies

1. atom $\mathtt{p}$ if $\mathtt{p} \in S$,
2. $\mathtt{not\ p}$ if $\mathtt{p} \notin S$,
3. $\mathtt{h_1} \vee \cdots \vee \mathtt{h_k}$ if for some $1 \leq i \leq k, \mathtt{h_i} \in S$
4. $\mathtt{b_1}, \ldots, \mathtt{b_n}$ if S satisfies every atom in it,
5. a rule r if, whenever S satisfies the body of r, it satisfies the head of r.

The solutions of an answer set program are defined as sets of ground atoms, called **answer sets**. To determine whether a candidate set S is an answer set of a program Π, the reduct of the program, Π^S, needs to be constructed. This is done in multiple steps: First, ground the program by replacing all variables with all possible ground constants mentioned in the program. Then, remove all rules containing $\mathtt{not\ p}$ such that $\mathtt{p} \in S$. Last, remove all remaining body atoms containing $\mathtt{not}$. S is an answer set of Π if it satisfies the rules of Π^S and is minimal (i.e. there is no proper subset of S satisfying the rules of Π^S).

The language contains more constructs to facilitate the modelling of complex problems. The main constructs include cardinality constraints, aggregates and optimisation statements.

A **cardinality constraint** is of the form

$$\mathtt{l}\{\mathtt{h_1}, \ldots, \mathtt{h_k}\}\mathtt{u} :- \mathtt{b_1}, \ldots, \mathtt{b_m}, \mathtt{not\ b_{m+1}}, \ldots, \mathtt{not\ b_n}$$

where $\mathtt{l}$ and $\mathtt{u}$ are integer values such that $l \leq u$. Whenever the body holds, between $\mathtt{l}$ and $\mathtt{u}$ atoms in the head of the rule must be included in the answer set of the program. By leaving out the bounding numbers in a cardinality constraint, any combination of atoms in that set can be included in an answer set. Cardinality constraints are also known as **choice rules**.

Aggregates perform operations on sets. For example, the expression $\#\mathtt{count}\{\mathtt{X} : \mathtt{p(X)}\}$ represents the number of elements in $\mathtt{p}$. Other operations include sums, maxima and minima. Aggregates are often used in conjunction with comparison operators to model complex relationships.

Optimisation statements instruct the answer set solver to find solutions that either minimise or maximise certain properties. They are written using directives such as $\#\mathtt{maximize}\{\mathtt{X} : \mathtt{p(X)}\}$. Rule bodies can also be annotated with weights, so-called weak constraints. The solver finds the optimal solution and ranks the answer sets based on these criteria.

s(ASP) (Marple et al. 2017) is an extension of the language, which provides solutions top-down in a goal-driven manner. Given a query, the system computes a partial answer set that contains it, bypassing the need to compute the entire answer set. Moreover, it does not need to ground the program, leading to performance improvements for some problems. A further extension is **s(CASP)** (Arias et al. 2018), which introduces constraints on variables, including over dense and unbounded domains.

ASP is a modelling language, meaning that problems are represented in a declarative way. To solve a problem, you typically model it in ASP and a solver calculates the possible solutions. One widely used solver is Clingo (Gebser et al. 2017). An answer set program can have multiple solutions and can represent problems up to the second level of the polynomial hierarchy (Law et al. 2018). ASP is also non-monotonic, meaning that adding new rules can invalidate previously held conclusions. This enables commonsense reasoning to be modelled through default rules, which usually hold unless disproven by new evidence.

Learning from answer sets

Rather than defining answer set programs by hand, the Learning from answer sets (LAS) task aims to automatically learn an answer set program from examples. This process, known as inductive learning, tries to find general rules that explain the given data. In this section, we define how to set up and solve a LAS task, using the definitions from Law et al. (2019).

Examples are represented in the form of weighted context-dependent partial interpretations (WCDPIs). A **WCDPI** is a tuple $e = \langle e_{id}, e_{pen}, e_{pi}, e_{ctx} \rangle$, where e_{id} is a unique identifier, e_{pen} is a penalty value, e_{pi} is a partial interpretation and e_{ctx} is the context. A partial interpretation e_{pi} consists of a pair of atom sets $\langle e^{inc}, e^{exc} \rangle$, called the inclusion and exclusion sets. The context is written in the form of an answer set program. A program P accepts a WCDPI e, iff there exists at least one answer set $I \in AS(P \cup e_{ctx})$, such that $e^{inc} \subseteq I$ and $e^{exc} \cap I = \emptyset$.

A **LAS task** is a tuple $T = \langle B, S_M, \langle E^+, E^- \rangle \rangle$, where B is the background knowledge, S_M is the hypothesis space and E^+, E^- are sets of positive and negative WCDPIs. B is simply represented as an answer set program and S_M is a set of ASP rules. The goal is to learn an optimal hypothesis $H \subseteq S_M$, such that $B \cup H$ accepts as many positive WCDPIs and as few negative WCDPIs as possible. The optimality of H is determined by summing up the penalties of negative/positive WCDPIs that are accepted/not accepted respectively. In addition, a penalty for the length of H is applied to encourage shorter hypotheses. Since it is infeasible to define S_M as a list of all possible rules that H could contain, it is declared using a mode bias in practice. The **mode bias** consists of declarations and other constructs that specify the hypothesis space.

Learning a hypothesis that defines concepts already observed in the examples is known as observational predicate learning (OPL). This is computationally easier than non-OPL tasks, which require learning concepts that are not directly observed (Law et al. 2021).

The two main LAS frameworks are ILASP (Law et al. 2020b) and FastLAS (Law et al. 2020a). ILASP is capable of learning full answer set programs. FastLAS is more scalable but also more restricted. It does not learn constructs such as choice rules or recursive rules and cannot invent new predicates. Both systems are purely symbolic and cannot natively integrate neural networks.

Neural networks

Neural networks are composed of multiple layers of nodes, which are connected by weighted vertices. They process an input by pushing it from one layer to the next, transforming it with linear and non-linear functions. The transformed data that is output from the last layer represents the result. A dataset of input-label pairs is used to learn a task. For each dataset example, the neural network output is compared with the label and a loss is calculated. The weights of the neural connections are changed with regards to the loss through the process of backpropagation (LeCun et al. 2015). Unlike symbolic methods, neural networks represent knowledge sub-symbolically via their structure and the values of the weights applied to connections.

CNNs. A class of neural networks for image classification are convolutional neural networks (CNNs). They contain specialist convolutional layers, which detect local features, and pooling layers, which merge the features into higher-level concepts. These operations are particularly suited for images, as the different convolutional functions act like filters and extract different properties from the input (LeCun et al. 2015).

Foundation models. Neural networks with billions of parameters that are trained on vast amounts of broad data are known as foundation models (Bommasani et al. 2022). They are capable of solving a wide range of problems through the use of **in-context learning**. The model learns how to solve a task from examples provided in the prompt as a natural language description. No weights are changed in this process, instead the model is only conditioned to utilise existing parameters. This type of adaptation is known as few-shot learning. When no examples are provided in the prompt, the model performs zero-shot learning. Performance can be improved by encouraging the model to break down its reasoning into steps, a process known as chain-of-though-prompting. **Finetuning** is used to adapt a model to task-specific data by changing its weights.

The two main types of foundation models are large language models (LLMs) and vision language models (VLMs). LLMs are based on the transformer architecture and process text through the mechanism of attention. VLMs combine an LLM with a vision encoder to process multimodal input in the form of text and images (Bordes et al. 2024).

Neurosymbolic ASP frameworks

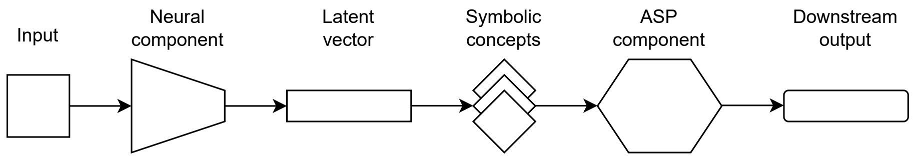


Figure 1. High-level depiction of inference through the components of neurosymbolic ASP frameworks.

There are a wide variety of frameworks which combine neural networks and ASP. In all approaches, the neural component processes raw inputs, while the ASP component performs logical reasoning to create an output, as illustrated in Figure 1. Combining neural and symbolic methods requires a translation between the latent vector representation of neural outputs and the symbolic concept representation of symbolic reasoners. We illustrate a typical inference procedure through such a neurosymbolic architecture with an example.

Example. In the MNIST Addition task, each input consists of two images of handwritten digits from the MNIST dataset (Deng 2012). Each downstream output is a single number, representing the sum of the two input numbers. In a neurosymbolic framework, the neural component can be a simple CNN which processes one image at a time and produces a latent vector of size 10. The ith entry in the vector represents the probability of the input being number i, for $i \in \{0, \ldots, 9\}$. By choosing the `argmax`, i.e. the index of the entry with the highest probability, each latent vector can be translated into a symbolic concept. The ASP component can include a rule for adding up the two symbolic concepts: $\texttt{result}(\texttt{Z}) :- \texttt{digit}(1, \texttt{X}), \texttt{digit}(2, \texttt{Y}), \texttt{Z} = \texttt{X} + \texttt{Y}$. The downstream output is the number $\texttt{Z}$ in $\texttt{result}(\texttt{Z})$.

Compared to fully neural methods, a neurosymbolic approach has the advantages of robustness and explainability. The ASP component provides a decision based on human-readable rules, in contrast to an opaque neural network, which uses layers of nodes and weights. Any conclusion output by the ASP component is also logically robust given these set of rules, which is not guaranteed with a neural network. However, the increased transparency comes at the cost of complexity.

First, the translation between the continuous vector space of neural networks and the discrete symbols and rules of ASP is not always trivial. Symbols have to be extracted from raw data, such as natural language text or images. Depending on the problem, the translation may involve an unknown number of concepts or complex perception tasks.

Second, providing a learning signal for the neural and/or symbolic component is difficult. In a traditional neural network task, the model is trained end-to-end using the input and labels. For neurosymbolic ASP frameworks, the neural network outputs latent concepts, for which labels are often unavailable. Instead, the learning signal comes from the downstream labels, which have to be propagated through the non-differentiable ASP component. In many tasks, different combinations of latent symbols can result in the same downstream output, providing a noisy learning signal to the neural network. Learning the ASP component itself is challenging as well, as it receives noisy inputs from the neural network.

The proposed frameworks in the literature are all structured differently and deal with their own set of challenges. They can be split up into four broad categories:

1. Frameworks with a pre-trained neural and hard-coded ASP component. Their main challenge lies in the translation of neural outputs into symbols.
2. Frameworks with a hard-coded ASP component that train a neural network. Their main challenge lies in the propagation of the downstream learning signal through the ASP component.
3. Frameworks with a pre-trained neural component that learn an answer set program. Their main challenge lies in the learning of ASP rules with noisy neural predictions.
4. Frameworks that learn the neural and ASP component jointly. Their main challenge is a combination of all the problems above.

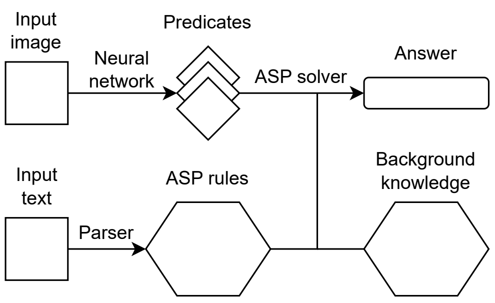
In this section, we discuss each category and illustrate the frameworks within it.

Pre-trained neural and hard-coded symbolic component

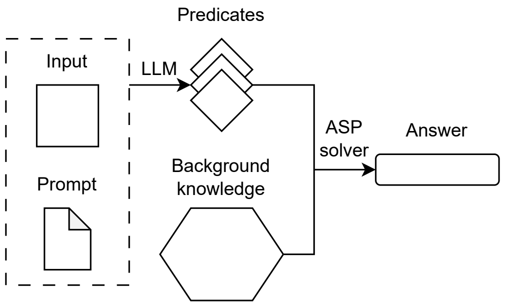
When both the neural and symbolic components are already given, the main focus lies on bridging the gap between them. Papers in this area tend to choose challenging tasks with natural language texts and complex images to demonstrate the usefulness of their framework. While earlier works use hand-crafted parsing pipelines, more recent papers experiment with LLMs.

Figure 2a depicts the general structure of frameworks for the problem of visual question answering (VQA). In VQA, the task involves answering questions about an image, such as "How many blue objects are in the scene?"

ASP-VQA and AQuA. Both the ASP-VQA (Eiter et al. 2022) and AQuA (Basu et al. 2020) frameworks use a YOLO network (Redmon and Farhadi 2018) to extract predicates from the image. YOLO is a neural network architecture that outputs bounding boxes for objects in an image. Each row in its output vector represents an object, and the columns correspond to class probabilities. Converting these neural vectors into symbols is done simply by picking the class with the highest probability. On top of that, ASP-VQA uses thresholding to select multiple likely classes per object and aggregates them into a choice rule. The ASP component can choose any of these top predictions, allowing room for error. As both approaches pre-train the YOLO network directly on given latent labels, no learning is occurring, just inference.



(a) High-level depiction of the inference procedure in ASP-VQA (Eiter et al. 2022), AQuA (Basu et al. 2020) and NSGRAPH (Bauer et al. 2025).



(b) High-level depiction of the inference procedure in [LLM]+ASP (Yang et al. 2023), STAR (Rajasekharan et al. 2023b) and LLMASP (Alviano et al. 2024).

Figure 2. High-level depictions of frameworks with pre-trained neural and hard-coded symbolic components.

To extract the query and knowledge from text questions, both frameworks use a parsing pipeline. ASP-VQA does not use the natural language text directly, but its functional representation, which the dataset provides. The functional representation is a structured format made up of function symbols, predicates and relations. The translation into ASP can therefore be done by a straightforward set of parsing rules. AQuA, on the other hand, parses the natural language text and converts it into ASP by utilising a part-of-speech tagger and dependency parser from CoreNLP (Manning et al. 2014). Both approaches also include extensive background knowledge of the task hard-coded in ASP. An ASP solver then calculates the answer by combining the extracted predicates and background knowledge.

By adding a symbolic layer on top of the YOLO network, the frameworks achieve the capability to answer complex queries. AQuA even exceeds human baseline performance on one of their datasets. The symbolic layer also adds robustness, as ASP-VQA reports good results even when the network is poorly trained.

Eiter et al. (2023) extend ASP-VQA with the ability to provide contrastive explanations. The enhanced framework is able to explain why the answer is P, rather than foil F, by showing how the input would need to change to yield F. This abduction problem is encoded ASP by spanning the search space with choice rules, adding constraints to ensure the answer contains the foil and adding weak constraints that prefer minimal changes. For example, a question might ask whether there are three purple objects to the left of the sphere in a given image of 3D objects. A contrastive explanation could include the answer *no* and the explanation that shifting the sphere 20 pixels to the right would change the answer to *yes*. The authors use the CLEVR dataset (Johnson et al. 2017) as a testbed, where the each input is a scene of 3D objects and a corresponding question. They expand the dataset with 20 questions specific to contrastive explanations and achieve an accuracy of 99%.

NSGRAPH. Similar techniques have been applied to graph problems in (Bauer et al. 2025). The input images are pictures of graphs inspired by metro maps and the input text contains questions like “How many stations are between A and B?”. To parse the input picture, NSGRAPH uses an optical graph recognition network for the nodes and edges, and an optical character recognition network for the label text. The questions are parsed using regular expressions, or LLMs for harder tasks. The dataset is based on graph images generated from CLEGR (Mack and Jefferson 2018), which the authors call CLEGR^V. They also present two further extensions: CLEGR$^+$ and CLEGR-HUMAN. The former adds reformulated questions by replacing words with synonyms and rephrasing sentences. The latter adds questions that have been hand-crafted by humans using an online survey. NSGRAPH achieves accuracies of 85% and 94% respectively.

Recently, LLMs have taken over the process of parsing natural language input. Their remarkable ability to produce structured language output from natural language makes them well suited for the task of extracting ASP facts from text. They also require much less hand-crafting than pipelines with taggers and parsers. Figure 2b illustrates the use of LLMs in bridging the gap between text and ASP predicates.

[LLM]+ASP. Yang et al. (2023) use LLMs for extracting ASP facts from natural language text puzzles in their [LLM]+ASP framework. As LLMs are general-purpose models, in-context learning is employed through examples of correct extractions in the prompt. The extracted facts are combined with hard-coded background knowledge and an ASP solver arrives at the answer. The background knowledge comes in the form of knowledge modules, which are written in a general way and therefore reusable for different datasets. For example, the *location* module includes ASP rules for calculating an object's location using offsets and is used for spatial reasoning, navigation and path-finding problems. The authors show that the framework can solve robot planning tasks that the LLM alone fails at, thereby enhancing its capabilities. Moreover, their method has uncovered errors in datasets such as StepGame (Shi et al. 2021), where over 10% of datapoints contain conflicting information. While neural-only methods learn to fit to the errors, [LLM]+ASP flags them for being inconsistent.

Nguyen et al. (2025) have built a framework dedicated to detecting such errors. They integrate LLMs with the explainable ASP solver XClingo (Cabalar et al. 2020) to find misleading information in the CLUTRR dataset (Sinha et al. 2019). The LLM parses atoms from the input text and Clingo finds the answer sets. If they contain conflicting information, e.g. `mother(theresa,darnell)` and `sister(theresa,darnell)`, then XClingo generates an explanation tree that traces back to the source of the inconsistency. Their experiments show that about 15% of examples in CLUTRR contain inconsistent or ambiguous information.

STAR. The STAR framework (Rajasekharan et al. 2023b) extracts predicates from language inputs using an LLM and reasons over them with ASP. The authors make use of both in-context learning and finetuning to improve performance. Unlike [LLM]+ASP, they reason with the s(CASP) variation, which is query-driven and more scalable. s(CASP) adds the ability to justify an answer in form of a proof tree, which enhances explainability compared to using an LLM directly. Hard-coded background knowledge is once again needed to represent the commonsense knowledge necessary for solving the problems.

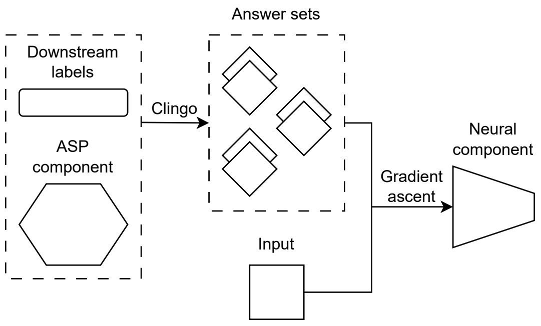
In further work, the authors use the STAR framework to create a neurosymbolic chatbot. In this scenario, the background knowledge is a conversational template that includes rules for staying on topic, gathering relevant information from the user and answering their questions. To create a natural flow of conversation, the LLM translates the ASP output into natural language again. Zeng et al. (2023) use this system to create a conversational agent called AutoConcierge using the LLM GPT3 (Brown et al. 2020). The aim is to provide restaurant recommendations based on nine relevant properties, e.g. location and price preferences, that are extracted from the user with natural language dialogues. Zeng et al. (2024) use the same system to create AutoCompanion, which is a social conversational bot for movies.

LLMASP. Alviano et al. (2024) use the YAML format to formalise the structure of prompts and background knowledge. The LLMASP system makes use of two hand-written YAML files to generate the prompt: The behaviour file includes generic instructions for the LLM to translate the input into ASP facts. The application file includes domain-specific knowledge that describes the context of the task and what kind of ASP facts should be extracted. Both files are combined into one prompt and the LLM extracts the relevant facts from the user input. An ASP solver then finds the answer, making use of the extracted facts and a hard-coded ASP knowledge base. Finally, the LLM translates the ASP facts into natural language to answer the user's questions. By providing the prompt structures in YAML, the system can be adapted to new domains in an efficient and rigorous manner.

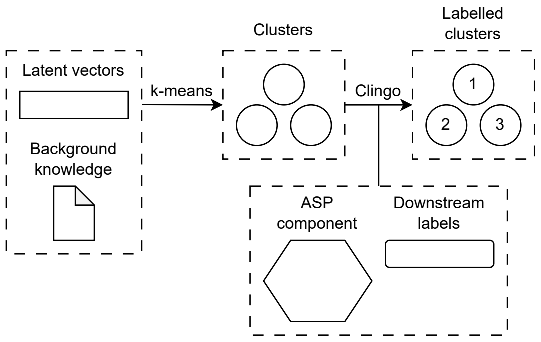
Overall, the frameworks in this section use symbolic components to improve the capabilities of neural networks and LLMs. They successfully apply parsers or LLM predictions to bridge the gap between raw data and ASP facts. Answers from such hybrid networks are more robust than from the neural networks alone and can even find errors in the dataset or include contrastive explanations. However, the scope has been limited to the interplay between neural and ASP methods, where no neurosymbolic learning is involved. The absence of automatic learning procedures also necessitates labour-intensive hard-coding, thereby limiting adaptability to new problems.

Neural training with hard-coded symbolic component

The frameworks in this section contain hand-written answer set programs and tackle the issue of training neural networks indirectly. This type of task is also referred to as neurosymbolic *reasoning*. The neural component must learn latent symbols without access to latent labels, instead relying on downstream labels. Thus, the main challenge lies in propagating the learning signal through the non-differentiable symbolic component to the neural network.



(a) High-level depiction of the training procedure for the neural component in NeurASP (Yang et al. 2020), SLASH (Skryagin et al. 2024b) and dPASP (Geh et al. 2024).



(b) High-level depiction of the cluster creation in Embed2Sym (Aspis et al. 2022).

Figure 3. High-level depictions of frameworks with hard-coded symbolic components that train the neural network.

Many frameworks in this section follow the same high-level procedure for training, which Figure 3a illustrates. For each example, they calculate all answer sets and use them as noisy labels for training the neural network. Where they differ is the loss functions they use and the structure of their neural components and learning algorithms.

NeurASP. The first framework of this kind is NeurASP (Yang et al. 2020), which trains a neural network given an answer set program and downstream labels. The neural component outputs a latent vector, which acts as a probability distribution over ASP concepts. A hard-coded ASP ruleset then calculates the downstream prediction using the most probable symbolic concepts from the neural network output. To propagate the learning signal through the symbolic program to the neural network, NeurASP first finds all answer sets that satisfy the downstream label. Each answer set contains a set of so-called neural atoms, which are the concepts that the neural component predicts. NeurASP calculates the probability of each answer set by multiplying the probabilities of the neural atoms in it. Finally, it calculates the gradient of the loss with respect to each neural output and performs gradient ascent. Given an entry in the neural latent vector, its gradient is increased for each answer set that contains it and decreased for each answer set that does not contain it, weighted by the probability of the answer set. In effect, the answer sets act as a weighted set of noisy latent labels.

Example. We revisit the MNIST Addition example from the beginning of this section to illustrate the training procedure. In NeurASP, the neural network predicts the value of a single digit and the sum operation is hard-coded in the ASP component. At training time, NeurASP calculates all answer sets for a given downstream label. For example, if the downstream label is 11, then the answer sets contain the following combinations of neural atoms: $\{(2,9),(3,8),(4,7),(5,6),(6,5),(7,4),(8,3),(9,2)\}$. Each answer set is assigned a probability based on neural network confidences. If the neural network is highly confident that the first number is 6 and the second number is 5, then the answer set $(6,5)$ will have a higher probability than, say, $(8,3)$. The gradient ascent operation will then increase the weights for predicting numbers 2 to 9, as they appear in the answer sets, and decrease the weights for 0 and 1, which do not appear in any answer set. The increases and decreases are weighted by the answer set probabilities.

Splitting up a task in this way alleviates pressure on the neural network. Instead of solving the entire task, it only has to learn latent concepts. Existing knowledge can then be utilised in the form of ASP rules to find the downstream solution.

SLASH. Skryagin et al. (2022) introduce SLASH, an extension of NeurASP that integrates more sophisticated probability estimations. In NeurASP, the perception component is a neural network and is only capable of estimating conditional probabilities for each symbol C given data X: $P(C|X)$. This is typically done using a Softmax function on its last layer. SLASH extends this notion with neural-probabilistic predicates (NPPs). NPPs can learn the probability distribution of the latent concepts, allowing SLASH to estimate $P(X|C)$ and $P(X,C)$ as well. Through density estimation, SLASH can also handle missing data points and regenerate them. In the paper, NPPs are realised using probabilistic circuits, but they can be replaced by any other component that estimates probabilities, including neural networks.

The scalability of SLASH is improved in Skryagin et al. (2024b), where the authors introduce a method called SAME to prune insignificant answer sets and speed up learning. As the neural predictions improve during training, SAME gradually eliminates symbolic latent concepts with low probabilities when generating answer sets. Over time, each epoch gradually speeds up, as the gradients are calculated using fewer and fewer answer sets. Further speed improvements come in the form of answer set networks (ASNs), which calculate answer sets by leveraging the GPU (Skryagin et al. 2024a). ASNs are answer set programs encoded into the form of a graph neural network, where nodes and edges represent the atoms and relations. This representation can compute answer sets in parallel on GPU nodes, unlike solvers like Clingo, which are CPU-bound. The authors use ASNs as a replacement for Clingo in SLASH and report speed improvements of two orders of magnitude for complex problems, allowing them to finetune LLMs with SLASH. However, ASNs require answer set programs to be tight, which require an exponential number of additional formulas in the worst case Lin and Zhao (2004).

dPASP. Geh et al. (2024) introduce a more powerful specification language with dPASP. It extends the capabilities of NeurASP and SLASH by introducing interval-valued facts and disjunctions that are annotated with probabilities. They implement two semantics for their framework: maxent and credal. The former assigns probabilities based on maximising entropy, while the latter is more conservative and assigns tighter bounds. The syntax of dPASP allows for the seamless integration of Python code and an interface between raw data and program constants. The learning function is based on a Lagrange multiplier derivation for gradient ascent and ends up being very similar to NeurASP's learning rule. It calculates the same terms as NeurASP, but multiplies them with weight factors $\frac{1}{m}$ and $1 - \frac{1}{m}$, where m is the number of possible atoms that the neural network output represents.

Embed2Sym. Unlike the previous frameworks, Embed2Sym (Aspis et al. 2022) fully pre-trains the neural component on the downstream labels. The neural network is structured to contain a separate perception and reasoning component, both of which are neural. The perception component processes the input and projects it into a latent dimension. The reasoning component takes the concatenated latent vectors and predicts the downstream output. This structure allows the entire neural network to be trained end-to-end with downstream labels, while producing a representation of latent concepts. To bridge the gap between latent vectors and symbolic concepts, the framework uses k-means clustering, as shown in Figure 3b. The number of clusters is equal to the number of values a latent concept can take and is hard-coded in the background knowledge. In the case of MNIST Addition, there are 10 clusters, one for each single-digit number. Matching each cluster with the correct concept label is done with Clingo using a hand-crafted answer set program. This program includes rules for reaching the downstream answer from latent concepts and chooses a cluster/concept matching that maximises the number of correct downstream predictions. In the MNIST Addition example, the answer set program would include rules for summing up the two latent concepts and assigning each digit the cluster that leads to the maximum number of correct sum predictions. At inference time, the framework uses the neural perception component to create latent vectors. It assigns each vector the symbolic label corresponding to the nearest cluster and then solves the task using the hard-coded ASP rules. The ASP component can be modified to solve new problems, such as subtraction, without retraining the neural network. This makes the model transferable to new domains, unlike a purely neural solution.

The latent embedding space of the neural component might not produce perfect clusters, leading to misclassified concepts. Rader and Russo (2023) alleviate this issue by extending the framework with active learning. They use the clusters to create a dataset of latent labels and finetune the perception component to predict latent concepts directly. For each example where the downstream prediction is correct, the cluster assignment is used as the the latent label. For examples with incorrect downstream predictions, an oracle provides *active* latent labels for the dataset. The extension improves the performance of the framework, enabling it to classify concepts more accurately than the clusters. As only a small number of datapoints need to be labelled and the network is not retrained from scratch, this extension is both data- and time-efficient.

Overall, there are two approaches to propagate the learning signal through a hard-coded ASP component. NeurASP, SLASH and dPASP use answer set calculations to enumerate all possible latent concepts for a downstream label. Embed2Sym trains a neural network end-to-end and uses clustering in the embedding space to generate latent labels. Either approach creates a more explainable framework and enables generalization to new tasks without the need of retraining, as only the ASP component has to be changed.

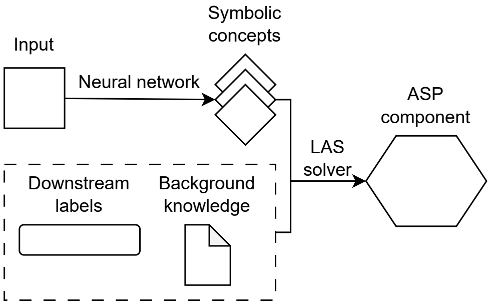
Symbolic learning with pre-trained neural component

Pre-training a neural component on latent labels alleviates the challenge of propagating the downstream learning signal back to the neural network. Papers in this section instead focus on translating neural outputs into symbols and learning an answer set program to solve the task. They use two main approaches for learning ASP rules: Either extracting symbolic concepts and then using a LAS solver, or generating rules directly with an LLM. Setting up a LAS task is not trivial, as the search space of possible rules is large and neural outputs are noisy. Generating rules with LLMs is challenging as well, because the free-form output has to be syntactically and semantically correct. In this section, we discuss the different strategies that have been proposed to overcome these issues.

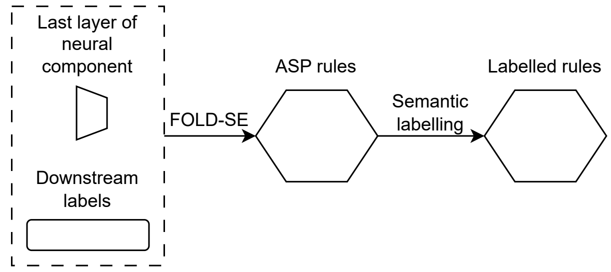
Figure 4a illustrates frameworks that deploy an off-the-shelf LAS solver like ILASP or FastLAS to find ASP rules. They differ in what neural methods they use to create the symbolic concepts for the LAS task.

FFNSL. The framework FFNSL (Cunnington et al. 2023a) uses a standard neural network that is pre-trained using latent labels. Their so-called data-to-knowledge generator translates the latent vector outputs of the neural component into ASP atoms by selecting the index of the maximum value in the vector. This translation is hard-coded, so it is known which vector entries correspond to which symbolic concepts. The symbolic component then learns an answer set program that solves the downstream task. It uses ILASP or FastLAS to find ASP rules based on the predicted symbolic concepts, the downstream labels and hard-coded background knowledge, which includes the search space for the possible rules. The paper investigates the effect of distributional shifts in the dataset, which cause the accuracy of the neural networks to plummet. However, the symbolic learning remains robust to noisy predictions and the generated rules outperform fully neural baselines.

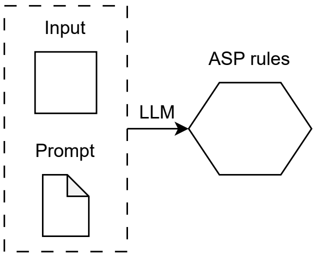
NeSyGPT. Instead of training a traditional neural network, NeSyGPT (Cunnington et al. 2024) extracts symbols from the data using a VLM. The framework feeds the input image into BLIP (Li et al. 2022) alongside a question designed to extract the latent concept. For example, the authors set the question "What number is this?" in the MNIST Addition task. For more complex perception tasks, such as extracting the suit and rank of a playing card, they additionally finetune BLIP with latent labels. Since there are no guarantees on the BLIP output, they use a text distance metric to map the VLM output to a predefined set of symbolic concepts. This interface between neural and symbolic components, i.e. the set of symbolic concepts and what questions to ask the VLM, is not necessarily manually engineered. The authors present a way to programmatically generate it using LLMs. After all examples have been converted into symbolic concepts, NeSyGPT learns ASP rules with ILASP.



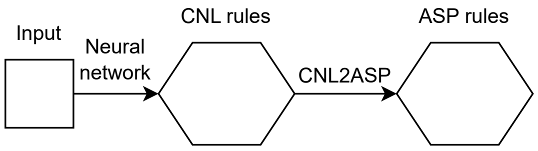
(a) High-level depiction of the symbolic learning procedure in FFNSL (Cunnington et al. 2023a), NeSyGPT (Cunnington et al. 2024), Embed2Rule (Aspis et al. 2024) and LLM2LAS (Kareem et al. 2025).



(b) High-level depiction of the symbolic learning procedure in NeSyFOLD (Padalkar et al. 2024).



(c) High-level depiction of the symbolic learning procedure in LLASP (Coppolillo et al. 2024).



(d) High-level depiction of the symbolic learning procedure in NL2ASP (Santana et al. 2024).

Figure 4. High-level depictions of symbolic learning procedures in frameworks with pre-trained neural components.

Embed2Rule. To reduce the number of calls to a VLM, Embed2Rule (Aspis et al. 2024) uses BLIP to label clusters rather than each individual example. It follows the same procedure as Embed2Sym to generate the clusters, which we illustrated in Figure 3b. Images from each cluster are then sampled and weakly labelled using BLIP. As BLIP might assign different labels to images in the same cluster, an optimisation algorithm finds the cluster-label assignment that maximises agreement with the BLIP labels. Lastly, the learned symbolic concepts are used to find rules with ILASP. Only requiring the VLM to label a few datapoints per cluster enhances the data and compute efficiency of this method.

LLM2LAS. The domain of LLM2LAS (Kareem et al. 2025) consists of story-based questions written in natural language. The framework utilises an LLM to generate predicates from the story using few-shot prompting. A parsing pipeline then converts these predicates into an event calculus representation depicting actions, effects and timepoints. Moreover, the pipeline creates mode biases and CDPIs for the LAS task automatically. Only a minimal amount of background knowledge, two lines of ASP, is hard-coded. The pipeline for creating predicates and mode declarations is also quite general, as long as the task can be represented in event calculus. This is the case for 13 out of 20 task types in the bAbI dataset (Weston et al. 2015) that the authors test LLM2LAS with.

In an updated version, Borroto Santana et al. (2025) replace the parsing pipeline with an LLM that generates mode biases for ILASP. This enables the framework to solve 15 rather than 13 tasks in bAbI. An additional two tasks can be solved with the help of extended background knowledge. For the rest, the hypothesis space remains too large for ILASP to find a solution.

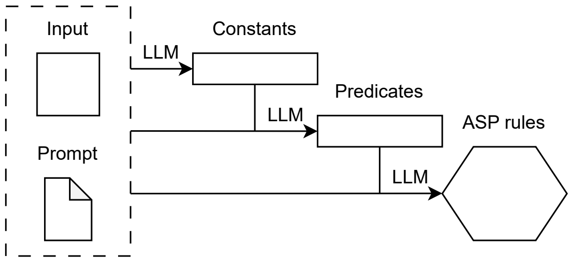
NeSyFOLD. The aim of NeSyFOLD (Padalkar et al. 2024) is not to solve a task, but to explain decision-making in CNNs. The CNN is pre-trained on downstream labels and NeSyFOLD turns its final layer into ASP rules, as Figure 4b illustrates. The rationale is that filters in the final layer tend to represent high-level concepts that can be formalised in logic. NeSyFOLD first binarises the last layer by thresholding the activation of each filter given an input, creating a tabular dataset. Then, the framework makes use of the FOLD-SE algorithm, which turns tabular data into default ASP rules (Wang and Gupta 2023). The resulting program approximates the decision-making of the last layer of the CNN by treating each filter as an atom that is used in the rule set. These atoms do not have human-readable names, which is why a semantic labelling step is necessary. An oracle, such as a human or foundation model, gives each atom a name by looking at the parts of the images that are activated by the filter that the atom represents. This results in a neuro-symbolic model that mostly maintains the predictive power of the CNN while being explainable and human-readable.

The authors have expanded the framework multiple times. NeSyFOLD-G (Padalkar et al. 2023) is a variant which groups similar kernels together before binarising them. This reduces the number of generated rules and therefore increases interpretability. NeSyBiCor (Padalkar et al. 2025) introduces the ability to remove biases in a CNN based on the rules extracted by NeSyFOLD. The user can tag undesired concepts in those rules that should not be used to make decisions. For example, the CNN might use the colour of the sky for predicting the type of road in an image, which is irrelevant. The framework then finetunes the CNN using a semantic similarity loss to push it away from making predictions with such undesired concepts. This process largely maintains the accuracy of the rule set and often reduces the number of rules.

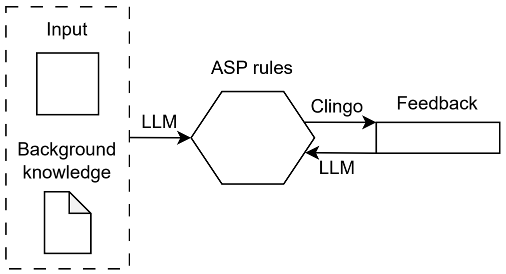
The remaining frameworks in this section utilise LLMs to create ASP rules. Their goal is to augment the reasoning capabilities of LLMs by encoding tasks in ASP instead of solving them directly. The remarkable ability of LLMs to produce structured languages such as Python code has been widely demonstrated in literature. However, as their training sets include much less ASP than Python, LLMs struggle to generate correct answer set programs from scratch. Different approaches therefore propose various methods to improve LLM capabilities.

LLASP. The conceptually simplest approach involves finetuning the LLM to output an ASP program directly, as is done in LLASP (Coppolillo et al. 2024). The authors first systematically evaluate the ASP capabilities of LLMs and find them to be inadequate in terms of syntactic and semantic correctness. To alleviate this problem, they perform supervised finetuning on lightweight LLMs using an ad-hoc dataset. This dataset consists of fundamental ASP programming patterns, such as constraints, joins, preferences or filtering. Despite being smaller in model size, LLASP is able to generate ASP rules in one shot, as illustrated in Figure 4c. On basic tasks, it achieves 89% semantic and syntactic accuracy on the ad-hoc dataset, outperforming much larger LLMs. When including combined problems, however, there are mixed results, indicating room for improvement in the field.

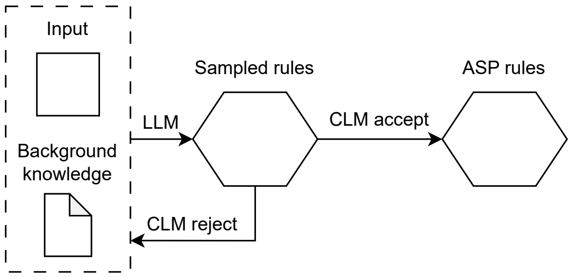
NL2ASP. Rather than outputting ASP directly, Santana et al. (2024) use an intermediate representation called controlled natural language (CNL). Figure 4d illustrates the two-step process. First, a neural network such as BART (Lewis et al. 2020) transforms natural language input into CNL. Second, the tool CNL2ASP (Caruso et al. 2023) translates the CNL sentences into ASP. The authors create a dataset to test out NL2ASP, consisting of ASP encodings taken from competitions and online resources and manually translated into CNL. To improve performance, they finetune the pre-trained neural network on CNL labels. NS2ASP consistently achieves BLEU scores over 0.9 and is able to produce 99% syntactically correct CNL statements with an F1 score of 0.93. The intermediate translation into CNL is an easier task than directly outputting ASP, as CNL is a higher-level language. Nevertheless, it still supports the main ASP constructs, such as facts, strong and weak constraints or choice rules.



(a) High-level depiction of the symbolic learning procedure in GPT-ASP (Ishay et al. 2023).



(b) High-level depiction of the symbolic learning procedure in DSPy-ASP (Wang et al. 2024) and LLM-ARC (Kalyanpur et al. 2024).



(c) High-level depiction of the symbolic learning procedure in CLM-ASP (Kaur et al. 2025).

Figure 5. More high-level depictions of symbolic learning procedures in frameworks with pre-trained neural components.

GPT-ASP. Ishay et al. (2023) devise a four-step method for converting natural language logic puzzles into answer set programs. Their framework GPT-ASP first generates constants, then predicates and then rules, as Figure 5a illustrates. At each step, the LLM has access to both the input and the ASP generated in the previous steps. The rule generation step is split up into two parts. First, the LLM generates choice rules, which increase the number of answer sets. Then, it creates constraints, which limit the number of answer sets again. This process is similar to how humans model problems in ASP. The pipeline allows you to spot and correct errors easily by inspecting the constructed constants, predicates and rules. This is not possible with LLM-only models that simply output the answer.

The capabilities of LLMs to generate ASP code can be further improved by introducing feedback loops. Two frameworks make use of this technique, which is shown in Figure 5b.

DSPy-ASP. In the DSPy-ASP framework (Wang et al. 2024), the LLM can revise the ASP rules for three iterations. First, it generates ASP predicates and queries, which are input to the Clingo solver together with predefined knowledge modules. Second, the LLM then revises the answer set program based on feedback from the solver, such as error messages. This process is repeated three times. While the LLM only generates predicates at first, it does have the ability to revise and generate new rules during the feedback loops. The authors use the DSPy Python framework (Khattab et al. 2024) to automate the prompt engineering process and show that adding feedback loops further improves task success. Compared to direct-prompting, the addition of ASP results in significant accuracy increases of up to 50% in spatial reasoning tasks.

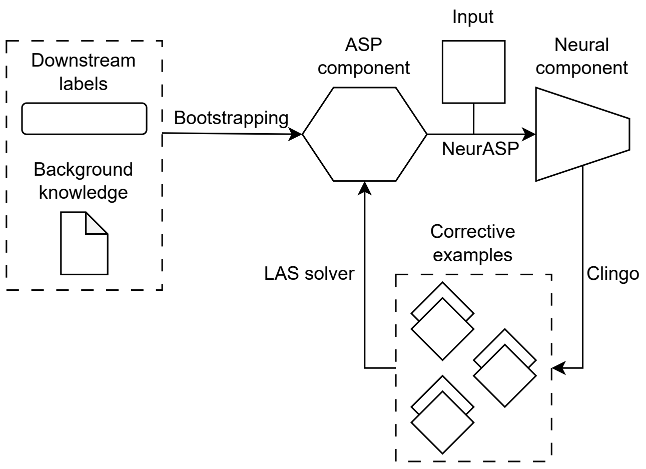
LLM-ARC. Kalyanpur et al. (2024) go further and use LLMs to generate tests in addition to ASP rules in their LLM-ARC framework. These tests are meant to verify the semantic correctness of the code. Just like the generated ASP rules, they are run through Clingo, which provides feedback through its error messages. The authors provide a simple schema for specifying tests, including mechanisms for checking that a proposition is true in any, all or no answer sets. The LLM can then correct the code and tests in an iterative manner, until everything compiles and all tests pass. The prompt includes few-shot examples of how to solve questions from the benchmark, including how to write tests and correct errors. Even though there are no guarantees that the generated tests are semantically sound, the authors show that they improve the correctness of the generated ASP rules in practice.

CLM-ASP. Kaur et al. (2025) apply conformal language modelling (CLM) to improve the capability of LLMs to produce answer set programs. CLM is a rejection sampling method with statistical guarantees that assesses the output of LLMs based on certain criteria. If the criteria are not met, CLM rejects the output and samples from the LLM again, as Figure 5c illustrates. The CLM check consists of two steps: First, all unacceptable samples that do not pass the admission function are filtered out. In the case of CLM-ASP, the admission function checks that the output is syntactically correct using Clingo. Second, the output is evaluated through the metrics of quality, diversity and confidence. The authors experiment with transition scores and $ROUGE_L$ metrics, as well as leveraging another LLM as the judge. The latter method involves finetuning an LLM to assess the quality of ASP and leads to better results than the other metrics. Finally, in-context learning is used to guide the creation of ASP. The prompt includes hand-written ASP rules that are needed to solve the given tasks.

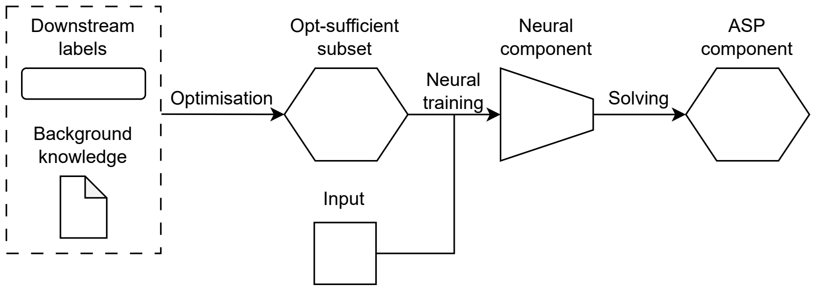
To sum up, there are two main approaches to learning ASP rules from raw data: Converting the data into symbolic examples to use with off-the-shelf solvers or generating rules directly. Both strands have been influenced by the rise of foundation models. In the former, VLMs act as the perception component, extracting predicates from images. In the latter, LLMs additionally act as the reasoning component, writing and improving ASP rules to solve a task using in-context learning. Foundation models have improved the scope and accuracy of neurosymbolic ASP methods. In turn, ASP has improved the logical capabilities of LLMs, which by themselves struggle with reasoning.

Joint learning of neural and symbolic components

Training the neural component and learning ASP rules at the same time is a very challenging task, because it resembles a chicken-and-egg problem. The neural network does not have latent labels to train with, as there is no answer set program that can generate them. And the ASP component does not have latent concepts to learn from, as the neural network is not trained yet. There are two papers in the literature that have tried to overcome these challenges without using any pre-trained components and we will discuss them in this section.



(a) High-level depiction of the learning procedure in NSIL (Cunnington et al. 2023b).



(b) High-level depiction of the learning procedure in NeuralFastLAS (Charalambous et al. 2023).

Figure 6. High-level depictions of frameworks that jointly learn the neural and ASP component.

NSIL. Cunnington et al. (2023b) address the chicken-and-egg problem in their NSIL framework by bootstrapping a hypothesis, as Figure 6a illustrates. The bootstrapping task takes only the downstream labels and background knowledge into account. It is set up using WCDPIs, each of which contains a downstream label in the inclusion set and choice rules for the neural atoms in the context. A LAS solver then finds ASP rules that cover as many WCDPIs as possible.

For tasks like MNIST Addition, bootstrapping works well. In the paper, the mode bias includes functions for addition, subtraction and multiplication. To maximise coverage of WCDPIs, the LAS solver would choose the addition function, because it is the only one that can cover all 20 downstream labels. A subtraction function of two positive digits cannot cover the labels 10 to 19, while a multiplication function cannot arrive at prime numbers like 13. For more complex tasks, the initial hypothesis might be wrong or incomplete, which is where the iterative nature of NSIL comes into play. The bootstrapped hypothesis is used to train the neural component with NeurASP. The freshly trained neural predictions are turned into corrective examples to learn a better hypothesis using FastLAS or ILASP. At this point, the loop starts again by training the neural network using the new hypothesis.

Splitting up the process into a neural and symbolic component allows NSIL to solve NP-complete tasks like the hitting set problem. It plays to the strengths of both paradigms, at the expense of a difficult learning regime.

NeuralFastLAS. Charalambous et al. (2023) introduce NeuralFastLAS, which learns ASP rules and trains a neural network jointly in one iteration, as shown in Figure 6b. First, the framework constructs a set of ASP rules that can prove the downstream labels for each example, given all possible choices of neural atoms. Using the background knowledge and constraints such as symmetry, this set of rules is pruned in the optimisation step to obtain the opt-sufficient subset. Crucially, the opt-sufficient subset is proven to contain the optimal symbolic solution. The answer sets stemming from the opt-sufficient subset are then used as noisy latent labels to train the neural network. Since there are many different rules in the opt-sufficient subset, the neural component has a second head that computes a posterior probability for each rule. After the network has been trained, an optimal hypothesis is found given the neural network predictions and rule posteriors. The paper proves the theoretic correctness of the method and provides conditions for the guaranteed convergence of the neural network. As the name suggests, the framework is modelled after FastLAS and thereby inherits its expressive power, which is limited to stratified programs.

The papers in this section tackle true neurosymbolic learning without any pre-training and very limited background knowledge. Both frameworks break down complex problems into a neural and symbolic part and try to solve both simultaneously - a very difficult task. They start the process by bootstrapping rules, either computing a single hypothesis or a space of possible hypotheses. While this approach works for simple examples, it tends to get stuck in local minima and is limited in its scalability. Much more research is needed to find algorithms that efficiently traverse the search space of possible rules while training neural components at the same time.

Analysis of neurosymbolic ASP

The field of neurosymbolic ASP contains a diverse set of frameworks, datasets and metrics. In this section, we aim to synthesize results across the landscape to identify the capabilities, strengths and limits of current methods. We start by evaluating the perception tasks used in benchmarks and conduct a comparative analysis of the performance of different frameworks. We find that neurosymbolic methods are often able to outperform fully neural methods not only in explainability, but also accuracy. However, we identify some barriers to progress, including simple perception inputs, a gap in challenging but solvable datasets, a limited capacity to generate ASP and scalability issues. Throughout this section, we propose ways forward to address these limitations.

Perception tasks

Synthetic images	
MNIST (Deng 2012)	Embed2Sym, NeurASP, SLASH, dPASP, FFNSL, NeSyGPT, Embed2Rule, NSIL, NeuralFastLAS
ShapeWorld (Kuhnle and Copestake 2017)	SLASH
CLEVR (Johnson et al. 2017)	ASP-VQA, AQuA, SLASH, NeSyGPT
CLEGR^V (Bauer et al. 2025)	NSGRAPH
Real-world images	
CIFAR-10 (Krizhevsky and Hinton 2009)	Embed2Sym
VQAR (Huang et al. 2021)	SLASH
Playing cards (Cunnington et al. 2023a)	FFNSL, NeSyGPT, Embed2Rule
PlantVillage (Hughes and Salathe 2016)	FFNSL
Indoor scenes (Quattoni and Torralba 2009)	FFNSL
PlantDoc (Singh et al. 2020)	NeSyGPT
Places (Zhou et al. 2018)	NeSyFOLD
German traffic signs (Stallkamp et al. 2012)	NeSyFOLD
Natural language text	
bAbI (Weston et al. 2015)	[LLM]+ASP, LLM2LAS
CLUTRR (Sinha et al. 2019)	[LLM]+ASP
gSCAN (Ruis et al. 2020)	[LLM]+ASP
StepGame (Shi et al. 2021)	[LLM]+ASP, DSPy-ASP, CLM-ASP
SpartQA (Mirzaee et al. 2021)	DSPy-ASP
Logic grid puzzles (Mitra and Baral 2015)	GPT-ASP
FOLIO (Han et al. 2024)	LLM-ARC
QuaRel (Tafjord et al. 2019)	STAR
ASP patterns (Coppolillo et al. 2024)	LLASP
NL2CNL (Santana et al. 2024)	NL2ASP

Table 1. Datasets used to train the perception components and the corresponding frameworks.

Table 1 provides a summary of all types of input that papers have used to test their frameworks. The datasets can be split up into three categories: synthetic images, real-world pictures, and natural language text. Within these categories, there are discrepancies about the difficulty of the perception task. In general, frameworks that incorporate more learning tend to use simpler perception tasks, while frameworks with pre-trained components can handle more realistic inputs.

Synthetic images. The MNIST dataset consists of 28x28 pixel greyscale images of handwritten digits (Deng 2012). It was created in 1994 and formed the basis for testing one of the first convolutional neural networks, LeNet-5, which already achieved 99% accuracy (Lecun et al. 1995). As such, it is considered one of the easiest perception datasets and even very small neural networks can learn to predict it perfectly. Nine of the 13 frameworks that take images as inputs are tested on MNIST images, despite its simplicity. The main reason is that the image classification forms only the first part of a more complex neurosymbolic task, such as addition or set membership. To convincingly demonstrate the real-life viability of these neurosymbolic frameworks, however, more realistic perception tasks are needed.

The ShapeWorld dataset is a step above MNIST, consisting of two-dimensional shapes with various orientations and colours against a white background (Kuhnle and Copestake 2017). The SLASH framework uses a variant with up to four shapes and trains a CNN to recognise properties of the objects (colour, shape, shade and size). The downstream label is the combination of all properties, for example $\texttt{has\_attributes}(\texttt{object1}, \texttt{red}, \texttt{circle}, \texttt{bright}, \texttt{small})$. Unlike in tasks like MNIST Addition, the downstream label therefore includes all four latent labels without obfuscating them. Therefore, the neural component is trained directly on the latent labels and the symbolic component only collects all latent attributes into one predicate. While the perception task with ShapeWorld is more difficult, the actual neurosymbolic task is easier than for MNIST tasks.

In the CLEVR dataset, the shapes are three-dimensional and can partially occlude each other (Johnson et al. 2017). Again, SLASH trains the neural component directly on the latent attributes of the objects. ASP-VQA and AQuA use a pre-trained YOLO network and do not perform any learning at all. NeSyGPT uses a VLM instead, which is pre-trained on a large corpus of general images. In addition, the authors finetune it with a small number of latent labels. Similarly, the CLEGR^V dataset generates images of graphs deterministically from their specifications. This results in limited variety and a straightforward perception task, which NSGRAPH solves using pre-trained graph recognition models.

In the category of synthetic images, only MNIST is truly used for neurosymbolic training of the perception component. Most MNIST tasks include just the downstream labels and therefore only provide noisy signals for the latent classification task. ShapeWorld, CLEVR and CLEGR^V, while embodying a more complex perception task, provide direct latent labels in all cases.

Real-world images. CIFAR-10 is a dataset of real-world images compressed to 32x32 pixels that depict one of ten object categories (Krizhevsky and Hinton 2009). Embed2Sym uses it for the CIFAR-10 Addition task, where each image category is arbitrarily assigned a number and the goal is to find the sum of two images. Just like in MNIST Addition, no latent labels are given, but the perception task is more difficult.

The Visual Question Answering and Reasoning (VQAR) dataset contains diverse real-world images with a much higher resolution than CIFAR-10 (Huang et al. 2021). But the SLASH framework uses a pre-trained network to find the bounding boxes of objects, sidestepping the difficult task of training the neural component.

The Playing cards dataset consists of photos of real playing cards with a resolution of 523x831 pixels (Cunnington et al. 2023a). It is used to learn the rules for determining the winner of card games. None of the three papers that use the dataset actually train the neural component from downstream labels.

FFNSL pre-trains the neural component on the latent playing card labels directly, while Embed2Rule and NeSyGPT use a VLM with latent finetuning.

The same is the case for the PlantVillage (Hughes and Salathe 2016) and Indoor scenes (Quattoni and Torralba 2009) datasets. While they contain real-world images of diseased crops and varied indoor rooms, FFNSL uses a pre-trained neural network. For the PlantDoc dataset, which contains images of diseased plants (Singh et al. 2020), NeSyGPT uses a VLM with latent finetuning.

The Places dataset contains images of indoor and outdoor scenes (Zhou et al. 2018) and is used by the NeSyFOLD and NeSyBiCor frameworks. The downstream labels represent scenes, while the latent concepts are objects in the image. NeSyFOLD first trains a CNN to predict the scene category and then extracts rules from the CNN's last layer. Each atom in those rules represents a (set of) filter activations, which roughly correspond to objects in the image. The names of the objects are provided through manual annotation of segmentation masks. Therefore, the framework is not able to classify latent concepts autonomously. These papers use the same techniques for the German traffic sign dataset which includes pictures of, shockingly, German traffic signs (Stallkamp et al. 2012).

While many of the datasets in this section display realistic scenes and objects, they are not used in a neurosymbolic training regime. Instead, almost all frameworks either pre-train or finetune the neural components on image labels directly, which amounts to a basic classification task. The notable exception is CIFAR-10, where latent symbols are extracted automatically without labels in the Embed2Sym framework.

Natural language text. The last category of datasets comprises collections of natural language texts, which are used by the LLM-based neurosymbolic frameworks. They all take the form of logical tasks or puzzles, making them well-suited for translating into ASP.

bAbI is a collection of questions that involve skills such as counting, path-finding or negation to solve. The questions are generated from a simulation of entities and actions, which are turned into natural language using a simple automated grammar (Weston et al. 2015). Therefore, the dataset is not too complex. CLUTRR poses the task of inferring family relations from short stories. It is more realistic than bAbI, as the stories were written by crowd-workers, who generated narratives from the generated kinship facts (Sinha et al. 2019). gSCAN represents a grid world in JSON format and asks natural language questions about how to achieve a goal. The questions are very direct instructions without much language variability and the answer comes in the form of a sequence of actions (Ruis et al. 2020). In all these datasets, [LLM]+ASP uses additional hand-written knowledge modules to solve the tasks. Thus, even though many of the tasks are complex, the framework requires substantial manual engineering. LLM2LAS manages to solve bAbI questions with more minimal background knowledge, but the pipeline only works for 15 out of the 20 tasks.

StepGame contains questions that require multi-hop spatial reasoning. It consists of descriptions of entities and their spatial relationships in a grid-based world and asks queries about their relative positions. These descriptions are generated automatically, but utilise different ways to describe spatial relations from a set of crowdsourced synonyms (Shi et al. 2021). Both [LLM]+ASP and DSPy-ASP make use of hand-written knowledge modules to solve the task, limiting their applicability in real-world scenarios. While CLM-ASP makes the LLM generate the entire answer set program, all the rules necessary are included in the prompt. This leads to the curious scenario that most errors are caused by the LLM incorrectly copying the rules from the prompt. A more complex benchmark is SpartQA, which consists of quantifier-based reasoning around blocks and objects, generated automatically (Mirzaee et al. 2021).

DSPy-ASP beats LLMs in terms of accuracy, but again at the expense of requiring manually specified ASP knowledge modules.

The logic grid puzzles dataset provides a set of categories, each containing an equal number of elements. The aim is to match elements based on clues given in the question. Since the dataset was compiled from a puzzle website, the questions are presumably human-made (Mitra and Baral 2015). GPT-ASP manages to achieve a high accuracy, unlike LLM-only methods, without relying on any hand-crafted ASP. Instead, all the necessary knowledge is encoded within the prompt, which contains instructions and a few solved examples from the dataset.

FOLIO consists of a set of premises and a conclusion. The task is to determine whether the conclusion is true, false or uncertain. It was created by experts in 2024 to challenge the state-of-the-art language models of the time and includes logically complex tasks written in natural language (Han et al. 2024). LLM-ARC manages to outperform LLM-only models without needing any hand-written ASP rules.

QuaRel is a set of commonsense physics questions based on properties like friction, speed or time. The questions were crowdsourced by asking people to come up with imaginative scenarios for the given relations (Tafjord et al. 2019). The STAR framework uses LLMs to extract predicates, while modelling the commonsense knowledge by hand in ASP. Both the ASP patterns (Coppolillo et al. 2024) and the NL2CNL (Santana et al. 2024) tasks are ad-hoc datasets created for their respective framework. The former represents classic ASP patterns, which are encoded as templates and can be combined for more complex questions. Patterns include guessing, constraints, joins, transitive closure, preferences and filtering. For the latter, the authors collected ASP problems from competitions, lecture notes and online resources. They hand-crafted their CNL and natural language representation to create the questions and labels. In both cases, the tasks represent standard toy problems, rather than real-world applications.

In summary, most natural language datasets were generated synthetically, with some using crowd-workers to enrich the questions. Since they are based on simulations or structured graphs, transforming them into ASP is more straightforward than with true natural language inputs. The notable exception is FOLIO, which was written by experts with the goal of providing a challenging and varied benchmark. It is therefore impressive that LLM-ARC achieves state-of-the-art results on it without using any hand-written ASP knowledge. Another potential issue is that all datasets other than FOLIO were created before the advent of LLMs and have been released publicly. As LLMs are trained on large amounts of publicly available data, it is possible that they have seen these datasets in their training procedure. For a fair analysis of LLM-based methods, authors should make sure to use datasets that the model could not have encountered before.

All in all, methods with more complex neurosymbolic requirements are limited to more rudimentary perception datasets. Only the two simplest visual datasets, MNIST and CIFAR-10, are used for training the neural component without latent labels. For any more advanced perception tasks, neurosymbolic frameworks either train their neural component directly or finetune a VLM with latent labels. The current limit for true neurosymbolic learning with ASP therefore lies with 32x32 pixel images with 10 categories. For textual inputs, most datasets are synthetically generated and the majority of frameworks need additional hard-coded ASP knowledge. Only LLM-ARC generates all ASP autonomously and has been tested on a challenging, real-world dataset with FOLIO. However, as LLMs are pre-trained on vast amounts of data, they might have encountered the same or similar problems during training.

Performance analysis

The experimental sections of the literature offer insights into the viability of proposed methods on different datasets. In this section, we consolidate the results from numerous papers and tasks and provide a comparative analysis. We find that neurosymbolic frameworks vary in terms of capabilities, but most are capable of achieving accuracies of 80-100% on benchmark tasks. Combining ASP with LLMs improves performance on logical tasks across the board, but the literature lacks comparisons between neurosymbolic methods.

Task / Framework	NeurASP	SLASH	dPASP	Embed2Sym	NSIL	NFastLAS	NeSyGPT	FFNSL	Embed2Rule	ASP-VQA	AQuA
MNIST arithmetic											
Add 2	98	99	94	98	98	–	–	–	–	–	–
Add 3	98	99	–	–	8	93	–	–	–	–	–
Add 4	98	99	–	94	–	–	–	–	–	–	–
Add 6	T/O	–	–	94	–	–	–	–	–	–	–
Add 8	T/O	–	–	92	–	–	–	–	–	–	–
Add 30	T/O	–	–	66	–	–	–	–	–	–	–
$a \times b + c$	–	–	–	–	87	97	–	–	–	–	–
Even9Plus	91	81	–	89	90	98	95	90	–	–	–
1k labels	90	85	–	72	88	–	95	88	–	–	–
Member (N entries)											
N=3	97	–	–	97	–	–	–	–	–	–	–
N=4	T/O	–	–	97	–	–	–	–	–	–	–
N=5	T/O	–	–	98	–	–	–	–	–	–	–
N=20	T/O	–	–	93	–	–	–	–	–	–	–
Hitting sets											
MNIST 5/4	–	50	–	–	100	–	–	–	99	–	–
MNIST 10/6	–	T/O	–	–	T/O	–	–	–	93	–	–
FashionMNIST	–	–	–	–	88	–	–	–	–	–	–
PlantDoc	64	–	–	77	–	–	99	98	–	–	–
Other tasks											
CIFAR10 add	T/O	–	–	85	–	–	–	–	–	–	–
ShapeWorld	–	85	–	–	–	–	–	–	–	–	–
Crop yield	–	–	–	–	–	–	–	100	–	–	–
Indoor scenes	–	–	–	–	–	–	–	100	–	–	–
Sudoku 4x4	–	98	–	–	50	–	–	100	98	–	–
Sudoku 9x9	–	T/O	–	–	50	–	–	100	78	–	–
Follow suit 4	25	25	–	–	25	–	100	100	86	–	–
Follow suit 10	10	10	–	17	10	–	100	100	–	–	–
CLEVR	99	90	–	–	–	–	99	–	–	97	94

Table 2. Performance comparison of different frameworks on various tasks. Numbers describe accuracy (%). NFastLAS = NeuralFastLAS.

Table 2 presents the accuracies of 11 frameworks on different tasks. We compiled these results from the original papers running their own framework, as well as papers running other frameworks for comparison.

MNIST arithmetic. Various tasks involve MNIST digits as inputs and perform arithmetic operations in the symbolic component. Addition is the most basic variation and involves calculating the sum of 2 up to 30 digits. Another variation includes multiplication of the first two digits added to the third. In Even9Plus, the result is the second value if the first value is even, or 9 plus the second value otherwise. The 1k labels task involves two-digit addition with training data limited to 1000 examples.

Eight frameworks report results on MNIST arithmetic tasks, demonstrating its role as an established benchmark for neurosymbolic ASP. As the first row in Table 2 shows, two-digit addition is a solved problem and all frameworks manage to achieve over 90% accuracy. Even9Plus and $a \times b + c$ are more challenging, with some frameworks dropping below 90%. Both NeurASP and SLASH do very well up until 4 digits in MNIST addition. From 6 digits onwards, NeurASP times out, whereas Embed2Sym manages to scale all the way up to 30. The reason lies in its structure: Embed2Sym trains a neural network end-to-end and then clusters the latent space. While the number of inputs increases, the number of clusters stays at 10, which remains manageable. However, the reliance on neural networks makes Embed2Sym less data-efficient, as it achieves the lowest accuracy among all frameworks tested on 1,000 labels.

Both NSIL and NeuralFastLAS learn the rules of the task while training their neural component simultaneously. This is a much harder challenge, but they manage to keep up with other frameworks on most tasks. The exception is 3 digit addition, where NSIL only achieves 8% accuracy. In general, NeuralFastLAS is superior, but it is more limited than NSIL in the rules it can learn. NeSyGPT and FFNSL are pre-trained on latent labels and therefore unsurprisingly perform well.

Member. Given a list of MNIST images and a symbolic digit, the member task involves stating whether the digit is included in the list of images. The downstream label is boolean and therefore only provides a sparse signal: If *false*, each image could represent any value except the given digit. If *true*, each image could represent any value, but at least one of them must be the given digit. Every added image in the list increases the number of possible latent labels by a factor of 10. NeurASP times out at a length of 4, where there are about $10^4 = 10,000$ answer sets per example. Embed2Sym scales much better again because it does not need to enumerate all latent possibilities. This illustrates the inherent scaling advantage of fully neural training well.

Hitting sets. In this task, you are given a universe of elements U and a set of sets S, where each $T \in S$ only contains elements from U. A hitting set $H \subseteq U$ contains elements such that for every $T \in S, T \cap H \neq \emptyset$. Given an integer $k \geq 1$, the hitting set task requires determining whether there exists a hitting set H with $|H| \leq k$ (Aspis et al. 2024). This task showcases the benefit of ASP's added expressivity, as it requires default negation to solve. Table 2 includes four variants: The first two use MNIST images as inputs with 5 elements in U and 4 input images, or 10 elements in U and 6 input images respectively, where $k = 2$. FashionMNIST contains greyscale images of clothing items and its hitting set task contains 5 elements in U with up to 4 sets per example and $k = 2$. Lastly, the PlantDoc hitting set task contain 38 elements in U with up to 5 sets per example and $k = 2$.

For this task, NeSyGPT, FFNSL and Embed2Rule fare much better than NeurASP, SLASH, Embed2Sym or NSIL. There is a clear divide between the former frameworks, which train their neural component, and the latter, which use pre-trained neural networks but learn the rules. This suggests that the ruleset itself is not too complicated, but does not provide an effective learning signal for the neural component. Therefore, hitting sets can serve as a good benchmark for further research.

Other tasks. CIFAR10 addition is reminiscent of two-digit MNIST addition, but the input images come from the CIFAR10 dataset. Each category is arbitrarily assigned a digit. The symbolic task in ShapeWorld amounts to predicting the attributes of each object, which is equivalent to a supervised classification problem. In crop yield prediction, the symbolic task defines the quality of the yield as poor, moderate or strong depending on the the crop's location, species and health. While the location is given, the species and health must be predicted from the input image. Indoor scene classification maps room classes (e.g. bathroom) into higher-level superclasses (e.g. home). The two Sudoku variants encode the grid validity task on boards of size 4x4 and 9x9. Given a sequence of MNIST images representing the board entries, the downstream label indicates whether the board is valid. In the follow suit game, the framework must determine the winner given 4 or 10 images of playing cards. The winner is the card that matches the suit of card 1 and has the highest rank among all matches. CLEVR involves determining the attributes of objects in a scene, such as like colour or material, and answering reasoning questions about them.

Despite the similarity to MNIST addition, NeurASP fails to learn CIFAR10 addition and times out. SLASH is able to achieve a high accuracy for ShapeWorld and FFNSL gets 100% on crop yield and indoor scenes. Note that FFNSL reports the accuracy of the learned hypothesis, rather than the downstream accuracy, which would take neural prediction errors into account. While NeurASP can learn 4x4 Sudoku grid validity, it times out for the larger 9x9 version. NSIL struggles in both cases and is unable to learn a valid hypothesis or train the neural network. Only FFNSL and Embed2Rule, which make use of pre-trained neural components, are able to achieve high accuracies on Sudoku.

A similar picture emerges for follow suit, where the predictions pf NeurASP, SLASH and NSIL all amount to random guessing. This happens because the learning signal from the downstream label is extremely limited. It only provides the index of the winner, and there can be millions of card combinations leading to the same winner in the 4-player version alone. In the 10-player version, there are up to $\binom{52}{10}$ possible latent label combinations per downstream label. Embed2Sym accomplishes a slightly better accuracy of 17% for Follow suit 10, but this is still much lower than for Member 20 or MNIST addition 30. That might be surprising, given that Follow Suit 10 has only has 10 inputs compared to 20 and 30 for the other tasks. The main difference is that playing cards have 52 unique values, one for each rank-suit combination. This requires Embed2Sym to create 52 clusters, as opposed to 10 for MNIST digits. Embed2Sym scales well with regard to the number of inputs, but struggles when the number of clusters increases. Finally, all frameworks achieve at least 90% accuracy on CLEVR across the board.

Table 3 summarises the results for the remaining frameworks, most of which are LLM-based. In terms of accuracy, there is a wider range of values, including challenging tasks like StepGame with 15 hops or Places with 10 classes. Only two datasets, bAbI and StepGame, have been used to test multiple frameworks. In all other cases, each paper has selected a different task on which to test their method. This illustrates a lack of agreed-upon baselines. Moreover, LLM-based frameworks only report results for their own method. Unlike the papers using traditional neural networks in Table 2, they do not run comparisons with other neurosymbolic frameworks.

Instead, papers report improvements compared to fully neural baselines. Combining LLMs and ASP yields better results than using LLMs by themselves, especially on logically challenging tasks. Yang et al. (2023) report that GPT3 alone can solve 80% of bAbI tasks with few-shot prompting, and 86% with chain-of-thought prompting. The neurosymbolic frameworks [LLM]+ASP and LLM2LAS both achieve 100% accuracy. In StepGame, the gap widens with an increasing number of hops. While GPT-3 achieves 32% fewer accuracy points for 1 reasoning hop, the difference grows to 57% at 10 hops. Wang et al.

Task / Framework	NS GRAPH	[LLM] +ASP	LLM 2LAS	DSPy-ASP	CLM-ASP	STAR	NeSy FOLD	GPT-ASP	LLM-ARC
CLEGR^V	73	–	–	–	–	–	–	–	–
bAbI	–	100	100	–	–	–	–	–	–
StepGame 1	–	93	–	94	80	–	–	–	–
StepGame 5	–	93	–	88	65	–	–	–	–
StepGame 10	–	88	–	80	–	–	–	–	–
StepGame 15	–	–	–	–	1	–	–	–	–
CLUTRR	–	91	–	–	–	–	–	–	–
gSCAN	–	100	–	–	–	–	–	–	–
SpartQA	–	–	–	70	–	–	–	–	
QuaRel	–	–	–	–	–	91	–	–	–
Places 2	–	–	–	–	–	–	92	–	–
Places 5	–	–	–	–	–	–	67	–	–
Places 10	–	–	–	–	–	–	44	–	–
German TS	–	–	–	–	–	–	78	–	–
Grid puzzles	–	–	–	–	–	–	–	92	–
FOLIO	–	–	–	–	–	–	–	–	88

Table 3. Performance comparison of different frameworks on various tasks. Numbers describe accuracy (%).

(2024) compare their DSPy-based approach with Deepseek, Llama3 and GPT-4.0 mini, reporting similar gaps for StepGame. For SpartQA, the results are closer together: Deepseek achieves 60%, LLama3 55% and GPT4.0 mini 56%.

Kaur et al. (2025) report 45%, 29% and 0% accuracy on StepGames 1, 5 and 15 respectively for a simple prompt-based approach with LLama3, much lower than CLM-ASP. The reason CLM-ASP fails on 15 hops is that the authors only calibrated the LLM with examples of up to 5 hops, indicating a lack of generalization capabilities in LLM-based ASP generation. Rajasekharan et al. (2023b) run tests with two versions of GPT3 with 7 billion and 175 billion parameters. The larger GPT model actually achieves the same accuracy as STAR on SpartQA. For the 7B version, however, STAR improves GPT3's accuracy form 77 to 86%. This suggests that neurosymbolic methods can help smaller models to match the performance of bigger models. The grid puzzles dataset poses the greatest challenge for vanilla LLMs, with accuracies of only 21% for GPT4, compared to the 92% achieved by GPT-ASP (Ishay et al. 2023). For FOLIO, GPT4 obtains 74% using chain-of-thought prompting (Kalyanpur et al. 2024).

Neurosymbolic methods offer advantages beyond increases in accuracy. For example, FFNSL maintains a better performance under distributional shifts than a neural network alone. Rajasekharan et al. (2023b) have conducted qualitative testing using real user input on their chatbots based on STAR. They conclude that STAR performs better than vanilla LLMs in metrics such as staying on topic or providing relevant responses. STAR can also generate justifications for each answer in form of a proof tree, a feature lacking in fully neural methods. Although NesyFOLD's rules result in a slight drop in performance compared to a CNN, they enable explanations of decisions and can correct biases.

Our comparative analysis has revealed the relative strengths and weaknesses of different approaches. Traditional neural networks scale better, but are less explainable and data-efficient than their neurosymbolic counterparts. For LLM-based approaches, the integration with ASP brings added value to reasoning tasks and outperforms chain-of-thought prompting. Common baselines like MNIST arithmetic pose little challenge to most frameworks, whereas harder tasks like follow suit lead to time-outs. Apart from hitting sets, there is a lack of challenging benchmarks where neurosymbolic frameworks achieve solid, but not near-perfect results.

ASP generation

ASP is able to express all NP-search problems and includes a variety of useful constructs such as disjunctions, choices and negation as failure to efficiently model statements Brewka et al. (2011). However, many neurosymbolic framework can only learn a subset of ASP. Furthermore, they often require extensive background knowledge and mode biases to limit the search space. Even LLMs, which can in theory generate any answer set program, are often limited to just producing predicates and have only been shown to generate deductive proofs rather than general knowledge. In this section, we will discuss these limitations, focussing on frameworks that learn at least part of an answer set program.

Expressivity limits. Predicates are the most basic form of ASP that a framework can produce and can be extracted directly from the input using a translation procedure. A few frameworks only generate predicates: ASP-VQA, AQuA and NSGRAPH utilise neural networks, while [LLM]+ASP, STAR and LLMASP take advantage of LLMs. The rest of the answer set program is either extracted using a fixed parsing pipeline or hard-coded in the form of knowledge modules. Some of these knowledge modules are general enough to be reusable for different tasks. Initially, the DSPy-ASP framework also generates just facts and relies on hard-coded ASP rules to form a program. However, it can change those rules and generate new ones in the iterative refinement stage based on Clingo feedback.

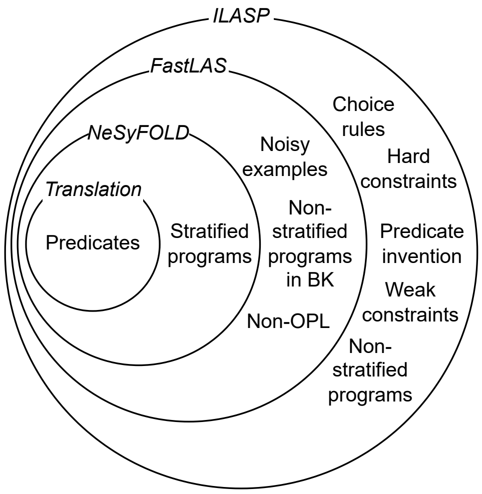


Figure 7. Levels of expressivity of different LAS frameworks.

Learning rules is a step up from predicates, but not all algorithms can generate rules that exploit the full expressivity of ASP. For example, NeSyFOLD uses the FOLD-SE algorithm, which can only learn default theories. These are equivalent to stratified answer set programs and cannot contain any cycles through negation. NeuralFastLAS is modelled after the first version of FastLAS, which is limited to observational predicate learning (OPL) and cannot learn recursive rules (Law et al. 2020a). Three further frameworks use FastLAS directly and are compatible with newer versions, which support non-OPL learning (Law et al. 2021): FFNSL, NSIL and NeSyGPT. Together with LLM2LAS and Embed2Rule, they also support ILASP for tasks that require higher expressive power. NSIL uses ILASP's capability to learn choice rules in the hitting set task. FFNSL, NeSyGPT and Embed2Rule require its predicate invention capability for the Follow suit task. Figure 7 provides a summary of the levels of expressivity and features that different frameworks can reach. Notably, FastLAS allows the use of non-stratified programs in the background knowledge, while ILASP supports it outright. ILASP can also learn higher-level ASP constructs such as hard and weak constraints, as well as choice rules.

Lastly, there are five frameworks that use LLMs to produce entire ASP rulesets: LLASP and CLM-ASP generate rules directly, with the latter using rejection sampling to pick the best program. GPT-ASP has a four step process and LLM-ARC generates both ASP rules and tests, refining them iteratively. NL2ASP generates sentences in CNL, which are then deterministically translated into ASP. As LLMs can output any combination of letters, they are in theory capable of producing any unrestricted answer set program, using all the syntactic constructs available in the language. This remains the case for NL2ASP, as CNL supports all standard constructs of ASP.

In summary, the expressive capabilities of generated answer set programs vary between frameworks. Seven frameworks only generate predicates, the lowest level of expressivity. NeuralFastLAS can learn rules, but is restricted to OPL tasks. Five frameworks use ILASP for generation, which gives them the ability to generate complex programs, including constructs such as negation as failure. Another five LLM-based frameworks generate rules directly, rather than just extracting predicates, allowing them to create rules of any level of expressivity in principle.

Background knowledge. Out of the 23 frameworks discussed in this survey, 11 hard-code the ASP component. Out of the remaining frameworks, six use FastLAS or ILASP and therefore make extensive use of background knowledge. FFNSL, NeSyGPT, Embed2Rule, NSIL and NeuralFastLAS use rule templates to restrict the search space and make the task tractable for FastLAS or ILASP.

We illustrate the extent of the background knowledge with the MNIST addition task that has been a running example throughout this survey. The following is an example of a typical rule template for this task, which we have adapted from NSIL:

```
num(0..18).
digit_type(0..9).
result(Y) :- digit(1,X0), digit(2,X1), solution(X0,X1,Y).
:- digit(1,X0), digit(2,X1), result(Y1), result(Y2), Y1 != Y2.
#modeh(solution(var(digit_type),var(digit_type),var(num))).
#modeb(var(num) = var(digit_type)).
#modeb(var(num) = var(digit_type) + var(digit_type)).
#maxv(3).
#bias("penalty(1, head(X)) :- in_head(X).").
#bias("penalty(1, body(X)) :- in_body(X).").
```

The first two lines restrict the domain of the labels (`num`) and digits (`digit_type`). The next line specifies that the result is the solution of the two input digits. The constraint ensures that there is only one result. The rest of the background knowledge consists of mode biases. `#modeh` specifies that the head of the learned rule must include a solution predicate with two digits and a number as its arguments. The `#modeb` lines tell the LAS solver that only a digit or the sum of two digits can be in the body of the learned rule. `#maxv(3)` restricts the LAS solver to a maximum of three variables in each rule. The last two lines specify that a penalty is given for each example that is not covered. The penalties instruct the LAS solver to find an optimal solution that covers as much of the data as possible.

As this example demonstrates, much of the ASP structure is still hand-crafted, even for a simple task like MNIST addition. NSIL also runs experiments with superfluous functions, such as subtraction or multiplication, to increase the hypothesis space. However, this still presents a vastly restricted search space compared to the universe of all possible functions. For more complex tasks, such as playing card games, even more information is needed to make the learning task tractable. Scaling up to real-world tasks would therefore require a substantial amount of manual engineering. This limits the usefulness of LAS solvers in the real world, because background knowledge is expensive to codify and sometimes unattainable. LLM2LAS is the only approach that automates some of the mode bias generation. An LLM generates the fluents in a sentence, which are then transformed into mode body and head atoms using a hard-coded pipeline. The use of LLMs to generate the ILASP search space instead of rules holds promise. It saves a lot of manual work and still results in a structured search for rules that provably fit the data.

Six frameworks generate 100% of their ASP rules themselves: NeSyFOLD, LLASP, NL2ASP, GPT-ASP, LLM-ARC and CLM-ASP. NeSyFOLD restricts the search space by only considering stratified ASP rules. As NL2ASP employs a traditional language model, it trains on the dataset using k-fold cross validation. All other papers use LLMs and guide the search through in-context learning. In these cases, some background knowledge is implicitly included in the prompt through solved examples.

Induction vs deduction. Learning answer set programs has traditionally been conducted through the lens of inductive logic programming (ILP). The goal of ILP is to learn general facts and rules from examples (Muggleton 1991). All neurosymbolic ASP frameworks in this survey that use traditional neural networks perform inductive learning. They generate one answer set program using multiple examples that models the entire dataset. The frameworks using LLMs, however, work differently. They generate a new answer set program for every example, with the aim of solving the puzzle in the question. The program models the natural language question in ASP and the solver then performs deductive inference to arrive at the solution. This is more akin to a translation and deduction task, rather than discovering new, general knowledge through induction.

LLMs in general struggle much more with inductive reasoning than deductive reasoning (Hua et al. 2025). Therefore, it remains to be seen if frameworks using LLMs are able to generate answer set programs inductively. More research and experiments are needed to develop this capability in neurosymbolic ASP.

Scalability

Table 4 summarizes the training times of different approaches on datasets. As the input sizes increase, NeurASP eventually times out on both addition and member tasks. NSIL calls NeurASP as part of its training loop, albeit with an improved implementation. This allows it to be quicker on tasks like MNIST

Task / Framework	NeurASP	SLASH	dPASP	Embed 2Sym	NSIL	NFast LAS	Embed 2Rule	ASP-VQA
MNIST Add 2	3m:13s	17s	20s	41s	–	–	–	–
MNIST Add 3	32m:26s	17s	–	–	18m:12s	48s	–	–
MNIST Add 4	56m:54s	1m:35s	–	1m:9s	–	–	–	–
MNIST Add 6	T/O (>4h)	–	–	1m:40s	–	–	–	–
MNIST Add 8	T/O (>4h)	–	–	2m:3s	–	–	–	–
MNIST Add 30	T/O (>4h)	–	–	7m:14s	–	–	–	–
Even9Plus	–	–	–	–	13m:42s	18s	–	–
Member 3	34m:4s	–	–	36s	–	–	–	–
Member 4	T/O (>4h)	–	–	43s	–	–	–	–
Member 5	T/O (>4h)	–	–	55s	–	–	–	–
Member 20	T/O (>4h)	–	–	2m:57s	–	–	–	–
Hitting set 5/4	–	2m:36s	–	–	68m:38s	–	15m:57s	–
Hitting set 10/6	–	T/O	–	–	T/O	–	2h:9m:14s	–
Sudoku 4x4	–	11m:31s	–	–	48m:8s	–	1h:0m:54s	–
Sudoku 9x9	–	T/O	–	–	12h:42m:13s	–	18h:6m:48s	–
Follow suit 4	–	T/O	–	–	T/O	–	3h:40m:19s	–
CLEVR	1h:0m:28s	–	–	–	–	–	–	89s

Table 4. Scalability comparison of different frameworks on various tasks. Numbers describe training time. NFastLAS = NeuralFastLAS.

add 3, despite learning the rules as well. However, on more complex tasks like hitting set 10/6 and follow suit, it times out too. A similar picture is painted by SLASH. It scales much better than NeurASP because it employs the SAME method, which prunes unlikely answer sets as the network grows more confident during training. Nevertheless, it times out once tasks get too complicated. Apart from implementation improvements, the training procedures of NeurASP, SLASH, dPASP and NSIL all boil down to the same computations: For each example, they enumerate all valid answer sets, calculate their probabilities and compute the gradients. This is feasible for tasks like MNIST addition with a few hundred possible answer sets. However, it is intractable for tasks like follow suit, which yields anywhere from 800,000 to 5 million answer sets.

Embed2Sym scales by far the best, as evidenced by Table 4. It can solve tasks where SLASH times out because it utilizes a fully neural training procedure. The gap to the symbolic space is crossed using clustering, a much faster method than computing all answer sets. Similarly, NeuralFastLAS vastly improves on NSIL by eliminating the need to use NeurASP and calculating a solution in a single iteration. Embed2Rule makes use of the same fully neural training procedure as Embed2Sym, allowing it to learn tasks where SLASH and NSIL fail. However, training times remain high for tasks like Sudoku 9x9, since it learns rules using ILASP.

Searching for solutions in the space of answer set programs is a difficult task to scale. For example, the complexity for ILASP to decide whether a hypothesis is an optimal inductive solution is Σ_2^P-complete (Law et al. 2018). This inherently limits the scalability of frameworks like Embed2Rule or NeSyGPT. FastLAS was created to be a more scalable LAS solver, but at a cost of features such as learning recursive rules or predicate invention (Law et al. 2021).

Such scalability issues can be bypassed with the use of foundation models, which do not calculate rules exactly, but rather predict the right words and symbols. ASP-VQA answers questions much faster than NeurASP on CLEVR through the use of an LLM. However, LLMs require a lot of resources to be trained in the first place, far surpassing the cost of training frameworks like NeurASP. Moreover, LLMs such as OpenAI's GPT models are proprietary and require dedicated server architecture to run, usually incurring a per-token cost. Unlike traditional methods, they cannot be run on local machines. Even smaller models like BLIP, which is used in NeSyGPT and Embed2Rule, require a lot of resources to finetune. Training them from scratch would be prohibitively expensive for researchers, which is why pre-trained models are used instead.

There is still a lot of work to be done to speed up ASP methods. While neural networks benefit from GPU parallelisation and very efficient Python frameworks, most ASP methods like Clingo, ILASP or NeurASP run on the CPU. There is progress on parallelisation, as with answer set networks, which can compute answer sets on the GPU (Skryagin et al. 2024a). More such methods are needed to cover all steps of neurosymbolic pipelines. Only with faster implementations, in addition to theoretical discoveries, can neurosymbolic ASP methods become more viable in real-world tasks.

Conclusion

In this survey, we have discussed the current literature on neurosymbolic ASP, highlighting the achievements and limitations of the field. We categorised the wide array of different frameworks according to which components are learned or hard-coded. Frameworks with fully pre-trained neural networks and hard-coded ASP components focus on the translation between the components. The addition ASP transforms decisions into transparent and verifiable processes and enables further features like contrastive explanations. When the neural components need to be trained, the main challenge lies in propagating the learning signal through the non-differentiable ASP component. Papers in this category either enumerate answer sets to serve as noisy labels or use clustering. There is room for further research into different methods of providing learning signals that scale better to the number of answer sets or clusters. The predictions of pre-trained neural networks can serve as noisy examples for LAS solvers, enabling rule learning from noisy data. Recent work has started utilizing LLMs to predict rules directly. Their results show that integrating ASP improves the reasoning capabilities of LLMs compared to traditional prompting. Lastly, jointly learning the neural and symbolic component serves as the most difficult challenge, as there are no reliable learning signals for either component at the start. Only two frameworks have been proposed so far and have made the first steps with simple problems.

The biggest challenge for neurosymbolic methods is scalability. The current limit for joint learning in perception tasks is CIFAR-10 and approaches using traditional neural networks time out when the task size increases. On reason lies in suboptimal implementations. Commonly-used frameworks like Clingo or ILASP do not have GPU support and cannot be parallelized easily. Code built on top of them, such as NeurASP, is often a research prototype and therefore not optimised. More efficient implementations are necessary for scaling up to larger tasks, but that is not enough Novel methods are needed for problems like efficiently traversing the search space of ASP rules and propagating learning signals through ASP components. A tighter integration between neural and symbolic representations could overcome some inefficiencies.

The need for extensive hard-coding also holds the field back. Hand-writing rules can be a very expensive processes and leads to poor generalizability. Even frameworks that learn rules needs to limit the search space with manually engineered rule templates. Recent work has explored the use of LLMs to automate rule generation, which represents a promising direction. LLMs can generate ASP rules directly and reduce the need for hard-coding, while ASP brings added value to their predictions in terms of reasoning ability and explainability. Further research is necessary to learn inductive knowledge with LLMs, for example by integrating them with LAS solvers. Another untapped research area is the combination ASP and foundation models for multi-modal inference, such as voice and text.

An opportunity for the field is development of more realistic datasets. The most popular benchmarks for traditional neurosymbolic methods are tasks based on MNIST images and frameworks routinely report accuracies of 80 or 90%. Most datasets for LLM-based methods use synthetically generated sentences and tasks with clear-cut symbolic representations. More realistic tasks are needed to demonstrate the applicability of neurosymbolic ASP in real-world settings. There is a need to create challenging but solvable tasks that can be widely adopted to measure different approaches against.

As neural models continue to struggle with complex logical reasoning, integrating efficient and search-based symbolic methods can achieve better performance, robustness and explainability. With its combination of expressiveness and readability, ASP is well-placed to fulfill this role. We have the opportunity to create trustworthy neurosymbolic frameworks that go beyond summarising existing information and towards discovering new knowledge.

Acknowledgements

This work was supported by the UKRI Centre for Doctoral Training in Safe and Trusted Artificial Intelligence [EP/S0233356/1].

References

Acharya K and Song H (2025) A comprehensive review of neuro-symbolic ai for robustness, uncertainty quantification, and intervenability. *Arabian Journal for Science and Engineering* DOI:10.1007/s13369-025-10887-3.

Agostinelli F, Panta R and Khandelwal V (2024) Specifying goals to deep neural networks with answer set programming. *Proceedings of the International Conference on Automated Planning and Scheduling* 34(1): 2–10. DOI:10.1609/icaps.v34i1.31454. URL `https://ojs.aaai.org/index.php/ICAPS/article/view/31454`.

Ahmed K, Teso S, Chang KW, den Broeck GV and Vergari A (2022) Semantic probabilistic layers for neuro-symbolic learning. In: *Neural Information Processing Systems*. pp. 29944–29959. URL `https://api.semanticscholar.org/CorpusID:249240063`.

Albilani M and Bouzeghoub A (2023) Guided hierarchical reinforcement learning for safe urban driving. In: *2023 IEEE 35th International Conference on Tools with Artificial Intelligence (ICTAI)*. pp. 746–753. DOI:10.1109/ICTAI59109.2023.00115.

Alviano M, Scudo FL, Grillo L and Reiners LAR (2024) Answer set programming and large language models interaction with yaml: Second report. In: *KoDis+CAKR+SYNERGY@KR*. pp. 4330–4338. URL `https://api.semanticscholar.org/CorpusID:274981259`.

Arias J, Carro M, Salazar E, Marple K and Gupta G (2018) Constraint answer set programming without grounding. *Theory and Practice of Logic Programming* 18(3–4): 337–354. DOI:10.1017/S1471068418000285.

Aspis Y, Albinhassan M, Lobo J and Russo A (2024) Embed2rule scalable neuro-symbolic learning via latent space weak-labelling. In: *Neural-Symbolic Learning and Reasoning: 18th International Conference, NeSy 2024, Barcelona, Spain, September 9–12, 2024, Proceedings, Part I*. Berlin, Heidelberg: Springer-Verlag. ISBN 978-3-031-71166-4, p. 195–218. DOI:10.1007/978-3-031-71167-1_11. URL https://doi.org/10.1007/978-3-031-71167-1_11.

Aspis Y, Broda K, Lobo J and Russo A (2022) Embed2Sym - Scalable Neuro-Symbolic Reasoning via Clustered Embeddings. In: *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning*. pp. 421–431. DOI:10.24963/kr.2022/44. URL https://doi.org/10.24963/kr.2022/44.

Badreddine S, d'Avila Garcez A, Serafini L and Spranger M (2022) Logic tensor networks. *Artificial Intelligence* 303: 103649. DOI:https://doi.org/10.1016/j.artint.2021.103649. URL https://www.sciencedirect.com/science/article/pii/S0004370221002009.

Barbara V, Guarascio M, Leone N, Manco G, Quarta A, Ricca F and Ritacco E (2023) Neuro-symbolic ai for compliance checking of electrical control panels. URL https://arxiv.org/abs/2305.10113.

Basu K, Shakerin F and Gupta G (2020) Aqua: Asp-based visual question answering. In: *Practical Aspects of Declarative Languages: 22nd International Symposium, PADL 2020, New Orleans, LA, USA, January 20–21, 2020, Proceedings*. Berlin, Heidelberg: Springer-Verlag. ISBN 978-3-030-39196-6, p. 57–72. DOI:10.1007/978-3-030-39197-3_4. URL https://doi.org/10.1007/978-3-030-39197-3_4.

Bauer JJ, Eiter T, Higuera Ruiz N and Oetsch J (2025) Visual graph question answering with asp and llms for language parsing. *Electronic Proceedings in Theoretical Computer Science* 416: 15–28. DOI:10.4204/eptcs.416.2. URL http://dx.doi.org/10.4204/EPTCS.416.2.

Baugh KG, Cingillioglu N and Russo A (2023) Neuro-symbolic rule learning in real-world classification tasks. URL https://arxiv.org/abs/2303.16674.

Belle V and Marcus G (2026) The future is neuro-symbolic: Where has it been, and where is it going? In: *Proceedings of the 40th AAAI Conference on Artificial Intelligence*, Proceedings of the AAAI Conference on Artificial Intelligence. AAAI Press, pp. 1–8. URL https://aaai.org/conference/aaai/aaai-26/. The 40th Annual AAAI Conference on Artificial Intelligence, AAAI-26 ; Conference date: 20-01-2026 Through 27-01-2026.

Bhuyan BP, Ramdane-Cherif A, Tomar R and Singh TP (2024) Neuro-symbolic artificial intelligence: a survey. *Neural Comput. Appl.* 36(21): 12809–12844. DOI:10.1007/s00521-024-09960-z. URL https://doi.org/10.1007/s00521-024-09960-z.

Bommasani R, Hudson DA, Adeli E, Altman R, Arora S, von Arx S, Bernstein MS, Bohg J, Bosselut A, Brunskill E, Brynjolfsson E, Buch S, Card D, Castellon R, Chatterji N, Chen A, Creel K, Davis JQ, Demszky D, Donahue C, Doumbouya M, Durmus E, Ermon S, Etchemendy J, Ethayarajh K, Fei-Fei L, Finn C, Gale T, Gillespie L, Goel K, Goodman N, Grossman S, Guha N, Hashimoto T, Henderson P, Hewitt J, Ho DE, Hong J, Hsu K, Huang J, Icard T, Jain S, Jurafsky D, Kalluri P, Karamcheti S, Keeling G, Khani F, Khattab O, Koh PW, Krass M, Krishna R, Kuditipudi R, Kumar A, Ladhak F, Lee M, Lee T, Leskovec J, Levent I, Li XL, Li X, Ma T, Malik A, Manning CD, Mirchandani S, Mitchell E, Munyikwa Z, Nair S, Narayan A, Narayanan D, Newman B, Nie A, Niebles JC, Nilforoshan H, Nyarko J, Ogut G, Orr L, Papadimitriou I, Park JS, Piech C, Portelance E, Potts C, Raghunathan A, Reich R, Ren H, Rong F, Roohani Y, Ruiz C, Ryan J, Ré C, Sadigh D, Sagawa S, Santhanam K, Shih A, Srinivasan K, Tamkin A, Taori R, Thomas AW, Tramèr F, Wang RE, Wang W, Wu B, Wu J, Wu Y, Xie SM, Yasunaga M, You J, Zaharia M, Zhang M, Zhang T, Zhang X, Zhang Y, Zheng L, Zhou K and Liang P (2022) On the opportunities and risks of foundation models. URL https://arxiv.org/abs/2108.07258.

Bordes F, Pang RY, Ajay A, Li AC, Bardes A, Petryk S, Mañas O, Lin Z, Mahmoud A, Jayaraman B, Ibrahim M, Hall M, Xiong Y, Lebensold J, Ross C, Jayakumar S, Guo C, Bouchacourt D, Al-Tahan H, Padthe K, Sharma V, Xu H, Tan XE, Richards M, Lavoie S, Astolfi P, Hemmat RA, Chen J, Tirumala K, Assouel R, Moayeri M, Talattof A, Chaudhuri K, Liu Z, Chen X, Garrido Q, Ullrich K, Agrawal A, Saenko K, Celikyilmaz A and Chandra V (2024) An introduction to vision-language modeling. URL https://arxiv.org/abs/2405.17247.

Borroto M, Ielo A, Mazzotta G and Ricca F (2025) Answer set programming and neurosymbolic ai: Applications and future perspectives (invited talk). In: Bruno P, Calimeri F, Cauteruccio F and Terracina G (eds.) *Hybrid Models for Coupling Deductive and Inductive Reasoning*. Cham: Springer Nature Switzerland. ISBN 978-3-031-89366-7, pp. 1–21.

Borroto Santana MA, GALLAGHER K, IELO A, KAREEM I, RICCA F and RUSSO A (2025) Question answering with llms and learning from answer sets. *Theory and Practice of Logic Programming* : 1–25DOI: 10.1017/S1471068425100343.

Bouneffouf D and Aggarwal CC (2022) Survey on applications of neurosymbolic artificial intelligence. URL https://arxiv.org/abs/2209.12618.

Brewka G, Eiter T and Truszczynski M (2011) Answer set programming at a glance. *Commun. ACM* 54: 92–103. DOI:10.1145/2043174.2043195.

Brown T, Mann B, Ryder N, Subbiah M, Kaplan JD, Dhariwal P, Neelakantan A, Shyam P, Sastry G, Askell A, Agarwal S, Herbert-Voss A, Krueger G, Henighan T, Child R, Ramesh A, Ziegler D, Wu J, Winter C, Hesse C, Chen M, Sigler E, Litwin M, Gray S, Chess B, Clark J, Berner C, McCandlish S, Radford A, Sutskever I and Amodei D (2020) Language models are few-shot learners. In: Larochelle H, Ranzato M, Hadsell R, Balcan M and Lin H (eds.) *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc., pp. 1877–1901. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Cabalar P, Fandinno J and Muñiz B (2020) A system for explainable answer set programming. *Electronic Proceedings in Theoretical Computer Science* 325: 124–136. DOI:10.4204/EPTCS.325.19.

Caruso S, Dodaro C, Maratea M, Mochi M and RICCIO F (2023) Cnl2asp: Converting controlled natural language sentences into asp. *Theory and Practice of Logic Programming* 24. DOI:10.1017/S1471068423000388.

Charalambous T, Aspis Y and Russo A (2023) Neuralfastlas: Fast logic-based learning from raw data. URL https://arxiv.org/abs/2310.05145.

Choi Y, Vergari A and den Broeck GV (2020) Probabilistic circuits: A unifying framework for tractable probabilistic models. URL https://api.semanticscholar.org/CorpusID:221997880.

Chu-Carroll J, Beck A, Burnham G, Melville DO, Nachman D, Özcan AE and Ferrucci D (2024) Beyond llms: Advancing the landscape of complex reasoning. URL https://arxiv.org/abs/2402.08064.

Cingillioglu N and Russo A (2021) pix2rule: End-to-end neuro-symbolic rule learning. *International Workshop on Neural-Symbolic Learning and Reasoning* URL https://api.semanticscholar.org/CorpusID:235422026.

Ciravegna G, Barbiero P, Giannini F, Gori M, Liò P, Maggini M and Melacci S (2023) Logic explained networks. *Artificial Intelligence* 314: 103822. DOI:https://doi.org/10.1016/j.artint.2022.103822. URL https://www.sciencedirect.com/science/article/pii/S000437022200162X.

Clocksin WF and Mellish C (1981) *Programming in Prolog*. Springer. URL https://api.semanticscholar.org/CorpusID:10092527.

Colelough BC and Regli W (2025) Neuro-symbolic ai in 2024: A systematic review. *ArXiv* abs/2501.05435. URL https://api.semanticscholar.org/CorpusID:274180938.

Coppolillo E, Calimeri F, Manco G, Perri S and Ricca F (2024) Llasp: fine-tuning large language models for answer set programming. In: *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning*, KR '24. ISBN 978-1-956792-05-8, pp. 834–844. DOI:10.24963/kr.2024/78. URL https://doi.org/10.24963/kr.2024/78.

Cunnington D, Law M, Lobo J and Russo A (2023a) Ffnsl: Feed-forward neural-symbolic learner. *Mach. Learn.* 112(2): 515–569. DOI:10.1007/s10994-022-06278-6. URL https://doi.org/10.1007/s10994-022-06278-6.

Cunnington D, Law M, Lobo J and Russo A (2023b) Neuro-symbolic learning of answer set programs from raw data.

Cunnington D, Law M, Lobo J and Russo A (2024) The role of foundation models in neuro-symbolic learning and reasoning. In: Besold TR, d'Avila Garcez A, Jimenez-Ruiz E, Confalonieri R, Madhyastha P and Wagner B (eds.) *Neural-Symbolic Learning and Reasoning*. Cham: Springer Nature Switzerland. ISBN 978-3-031-71167-1, pp. 84–100.

Dai WZ and Muggleton S (2021) Abductive knowledge induction from raw data. In: Zhou ZH (ed.) *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. International Joint Conferences on Artificial Intelligence Organization, pp. 1845–1851. DOI:10.24963/ijcai.2021/254. URL https://doi.org/10.24963/ijcai.2021/254. Main Track.

Dai WZ, Xu Q, Yu Y and Zhou ZH (2019) Bridging machine learning and logical reasoning by abductive learning. In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., pp. 2815–2826.

De Raedt L, Kimmig A and 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. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., p. 2468–2473.

De Smet L, Zuidberg Dos Martires P, Manhaeve R, Marra G, Kimmig A and De Raedt L (2023) Neural probabilistic logic programming in discrete-continuous domains. In: Evans RJ and Shpitser I (eds.) *Proceedings of the Thirty-Ninth Conference on Uncertainty in Artificial Intelligence*, *Proceedings of Machine Learning Research*, volume 216. PMLR, pp. 529–538. URL https://proceedings.mlr.press/v216/de-smet23a.html.

Deng L (2012) The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* 29(6): 141–142.

Dong H, Mao J, Lin T, Wang C, Li L and Zhou D (2019) Neural logic machines. In: *International Conference on Learning Representations*. pp. 2574–2595. URL https://openreview.net/forum?id=B1xY-hRctX.

Eiter T, Geibinger T, Higuera N and Oetsch J (2023) A logic-based approach to contrastive explainability for neurosymbolic visual question answering. In: Elkind E (ed.) *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*. International Joint Conferences on Artificial Intelligence Organization, pp. 3668–3676. DOI:10.24963/ijcai.2023/408. URL https://doi.org/10.24963/ijcai.2023/408. Main Track.

Eiter T, Higuera N, Oetsch J and Pritz M (2022) A neuro-symbolic asp pipeline for visual question answering. URL https://arxiv.org/abs/2205.07548.

Evans R, Bošnjak M, Buesing L, Ellis K, Pfau D, Kohli P and Sergot M (2021) Making sense of raw input. *Artificial Intelligence* 299: 103521. DOI:https://doi.org/10.1016/j.artint.2021.103521. URL https://www.sciencedirect.com/science/article/pii/S0004370221000722.

Farquhar S, Kossen J, Kuhn L and Gal Y (2024) Detecting hallucinations in large language models using semantic entropy. *Nature* 630: 625–630. URL https://api.semanticscholar.org/CorpusID:270615909.

Furelos-Blanco D, Law M, Jonsson A, Broda K and Russo A (2021) Induction and exploitation of subgoal automata for reinforcement learning. *J. Artif. Int. Res.* 70: 1031–1116. DOI:10.1613/jair.1.12372. URL https://doi.org/10.1613/jair.1.12372.

Garcez A and Lamb L (2023) Neurosymbolic ai: the 3rd wave. *Artificial Intelligence Review* : 1–20DOI:10.1007/s10462-023-10448-w.

Gebser M, Kaminski R, Kaufmann B and Schaub T (2017) Multi-shot ASP solving with clingo. *CoRR* abs/1705.09811.

Geh RL, Gonçalves J, Silveira IC, Mauá DD and Cozman FG (2024) dPASP: A Probabilistic Logic Programming Environment For Neurosymbolic Learning and Reasoning. In: *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning*. pp. 731–742. DOI:10.24963/kr.2024/69. URL https://doi.org/10.24963/kr.2024/69.

Gelfond M and Kahl Y (2014) *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*. Cambridge University Press.

Gibaut W, Pereira L, Grassiotto F, Osorio A, Gadioli E, Munoz A, Gomes S and Santos Cd (2023) Neurosymbolic ai and its taxonomy: a survey. *ArXiv* DOI:10.48550/ARXIV.2305.08876. URL https://arxiv.org/abs/2305.08876.

Han S, Schoelkopf H, Zhao Y, Qi Z, Riddell M, Zhou W, Coady J, Peng D, Qiao Y, Benson L, Sun L, Wardle-Solano A, Szabó H, Zubova E, Burtell M, Fan J, Liu Y, Wong B, Sailor M, Ni A, Nan L, Kasai J, Yu T, Zhang R, Fabbri A, Kryscinski WM, Yavuz S, Liu Y, Lin XV, Joty S, Zhou Y, Xiong C, Ying R, Cohan A and Radev D (2024) FOLIO: Natural language reasoning with first-order logic. In: Al-Onaizan Y, Bansal M and Chen YN (eds.) *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Miami, Florida, USA: Association for Computational Linguistics, pp. 22017–22031. DOI:10.18653/v1/2024.emnlp-main.1229. URL https://aclanthology.org/2024.emnlp-main.1229/.

Han Z, Cai LW, Huang YX, Wei B, Wang W and Yin Y (2023) Abductive subconcept learning. *Science China Information Sciences* 66. DOI:10.1007/s11432-020-3569-0.

Hitzler P, Eberhart A, Ebrahimi M, Sarker MK and Zhou L (2022) Neuro-symbolic approaches in artificial intelligence. *National Science Review* 9(6). DOI:10.1093/nsr/nwac035. URL https://doi.org/10.1093/nsr/nwac035.

Hitzler P and Sarker MK (2021) *Neuro-Symbolic Artificial Intelligence: The State of the Art*. IOS Press. URL https://api.semanticscholar.org/CorpusID:274983612.

Hua W, Sun F, Pan L, Jardine A and Wang WY (2025) Inductionbench: LLMs fail in the simplest complexity class. *Workshop on Reasoning and Planning for Large Language Models* URL https://openreview.net/forum?id=brw11PScQM.

Huang J, Li Z, Chen B, Samel K, Naik M, Song L and Si X (2021) Scallop: From probabilistic deductive databases to scalable differentiable reasoning. In: Ranzato M, Beygelzimer A, Dauphin Y, Liang P and Vaughan JW (eds.) *Advances in Neural Information Processing Systems*, volume 34. Curran Associates, Inc., pp. 25134–25145. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/d367eef13f90793bd8121e2f675f0dc2-Paper.pdf.

Hughes DP and Salathe M (2016) An open access repository of images on plant health to enable the development of mobile disease diagnostics. URL https://arxiv.org/abs/1511.08060.

Ishay A and Lee J (2025) Llm+al: bridging large language models and action languages for complex reasoning about actions. In: *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'25/IAAI'25/EAAI'25. AAAI Press. ISBN 978-1-57735-897-8, pp. 24212–24220. DOI:10.1609/aaai.v39i23.34597. URL https://doi.org/10.1609/aaai.v39i23.34597.

Ishay A, Yang Z and Lee J (2023) Leveraging large language models to generate answer set programs. In: *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning*, KR '23. ISBN 978-1-956792-02-7, pp. 374–383. DOI:10.24963/kr.2023/37. URL https://doi.org/10.24963/kr.2023/37.

Jin Y, Liu J, Luo Z, Peng Y, Qin Z, Dai WZ and Ding YX (2025) Pre-training meta-rule selection policy for visual generative abductive learning. *International Joint Conference on Learning and Reasoning 2024* : 163–180DOI: 10.1007/978-3-032-09087-4_11.

John-Mathews JM (2021) Critical empirical study on black-box explanations in ai. URL https://arxiv.org/abs/2109.15067.

Johnson J, Hariharan B, van der Maaten L, Fei-Fei L, Zitnick CL and Girshick R (2017) Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. pp. 1988–1997. DOI:10.1109/CVPR.2017.215.

Kairgeldin R and Carreira-Perpiñán MA (2025) Neurosymbolic models based on hybrids of convolutional neural networks and decision trees. In: H Gilpin L, Giunchiglia E, Hitzler P and van Krieken E (eds.) *Proceedings of The 19th International Conference on Neurosymbolic Learning and Reasoning*, *Proceedings of Machine Learning Research*, volume 284. PMLR, pp. 796–813. URL https://proceedings.mlr.press/v284/kairgeldin25a.html.

Kalyanpur A, Saravanakumar KK, Barres V, Chu-Carroll J, Melville D and Ferrucci D (2024) Llm-arc: Enhancing llms with an automated reasoning critic. URL https://arxiv.org/abs/2406.17663.

Kareem I, Gallagher K, Borroto M, Ricca F and Russo A (2025) Using learning from answer sets for robust question answering with llm. In: Dodaro C, Gupta G and Martinez MV (eds.) *Logic Programming and Nonmonotonic Reasoning*. Cham: Springer Nature Switzerland. ISBN 978-3-031-74209-5, pp. 112–125.

Kaur N, McPheat L, Russo A, Cohn AG and Madhyastha P (2025) An empirical study of conformal prediction in llm with asp scaffolds for robust reasoning. URL https://arxiv.org/abs/2503.05439.

Khattab O, Singhvi A, Maheshwari P, Zhang Z, Santhanam K, A SV, Haq S, Sharma A, Joshi TT, Moazam H, Miller H, Zaharia M and Potts C (2024) DSPy: Compiling declarative language model calls into state-of-the-art pipelines. *The Twelfth International Conference on Learning Representations* URL https://openreview.net/forum?id=sY5N0zY5Od.

Krizhevsky A and Hinton G (2009) Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Kuhnle A and Copestake A (2017) Shapeworld - a new test methodology for multimodal language understanding. URL https://arxiv.org/abs/1704.04517.

Lamb LC, d'Avila Garcez A, Gori M, Prates MO, Avelar PH and Vardi MY (2021) Graph neural networks meet neural-symbolic computing: a survey and perspective. In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, IJCAI'20. ISBN 9780999241165, pp. 4877–4884.

Law M, Russo A, Bertino E, Broda K and Lobo J (2020a) Fastlas: Scalable inductive logic programming incorporating domain-specific optimisation criteria. *Proceedings of the AAAI Conference on Artificial Intelligence* 34(03): 2877–2885. DOI:10.1609/aaai.v34i03.5678. URL https://ojs.aaai.org/index.php/AAAI/article/view/5678.

Law M, Russo A and Broda K (2018) The complexity and generality of learning answer set programs. *Artificial Intelligence* 259: 110–146. DOI:https://doi.org/10.1016/j.artint.2018.03.005. URL https://www.sciencedirect.com/science/article/pii/S000437021830105X.

Law M, Russo A and Broda K (2019) Logic-based learning of answer set programs. *Reasoning Web. Explainable Artificial Intelligence: 15th International Summer School 2019, Bolzano, Italy, September 20–24, 2019, Tutorial Lectures* : 196–231DOI:10.1007/978-3-030-31423-1_6. URL https://doi.org/10.1007/978-3-030-31423-1_6.

Law M, Russo A and Broda K (2020b) The ilasp system for inductive learning of answer set programs.

Law M, Russo A, Broda K and Bertino E (2021) Scalable non-observational predicate learning in asp. In: Zhou ZH (ed.) *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. International Joint Conferences on Artificial Intelligence Organization, pp. 1936–1943. DOI:10.24963/ijcai.2021/267. URL https://doi.org/10.24963/ijcai.2021/267. Main Track.

LeCun Y, Bengio Y and Hinton G (2015) Deep learning. *Nature* 521: 436–44. DOI:10.1038/nature14539.

Lecun Y, Jackel L, Bottou L, Cortes C, Denker J, Drucker H, Guyon I, Muller U, Sackinger E, Simard P and Vapnik V (1995) Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural networks* : 261–276.

Leonetti M, Iocchi L and Stone P (2016) A synthesis of automated planning and reinforcement learning for efficient, robust decision-making. *Artificial Intelligence* 241: 103–130. DOI:https://doi.org/10.1016/j.artint.2016.07.004. URL https://www.sciencedirect.com/science/article/pii/S0004370216300819.

Lewis M, Liu Y, Goyal N, Ghazvininejad M, Mohamed A, Levy O, Stoyanov V and Zettlemoyer L (2020) BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In: Jurafsky D, Chai J, Schluter N and Tetreault J (eds.) *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, pp. 7871–7880. DOI:10.18653/v1/2020.acl-main.703. URL https://aclanthology.org/2020.acl-main.703/.

Li J, Li D, Xiong C and Hoi S (2022) BLIP: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In: Chaudhuri K, Jegelka S, Song L, Szepesvari C, Niu G and Sabato S (eds.) *Proceedings of the 39th International Conference on Machine Learning*, *Proceedings of Machine Learning Research*, volume 162. PMLR, pp. 12888–12900. URL https://proceedings.mlr.press/v162/li22n.html.

Lifschitz V (2019) *Answer Set Programming*. 1st edition. Springer Publishing Company, Incorporated. ISBN 3030246574.

Lin F and Zhao Y (2004) Assat: computing answer sets of a logic program by sat solvers. *Artificial Intelligence* 157(1): 115–137. DOI:https://doi.org/10.1016/j.artint.2004.04.004. URL https://www.sciencedirect.com/science/article/pii/S0004370204000578. Nonmonotonic Reasoning.

Liu B, Jiang Y, Zhang X, Liu Q, Zhang S, Biswas J and Stone P (2023a) Llm+p: Empowering large language models with optimal planning proficiency. URL https://arxiv.org/abs/2304.11477.

Liu Y, Han T, Ma S, Zhang J, Yang Y, Tian J, He H, Li A, He M, Liu Z, Wu Z, Zhu D, Li X, Qiang N, Shen D, Liu T and Ge B (2023b) Summary of chatgpt/gpt-4 research and perspective towards the future of large language models.

Mack D and Jefferson A (2018) Clevr graph: A dataset for graph question answering. URL github.com/Octavian-ai/clevr-graph.

Manhaeve R, Dumančić S, Kimmig A, Demeester T and De Raedt L (2021) Neural probabilistic logic programming in deepproblog. *Artificial Intelligence* 298: 103504. DOI:https://doi.org/10.1016/j.artint.2021.103504. URL https://www.sciencedirect.com/science/article/pii/S0004370221000552.

Manhaeve R, Giannini F, Ali M, Azzolini D, Bizzarri A, Borghesi A, Bortolotti S, De Raedt L, Dhami D, Diligenti M, Dumančić S, Faltings B, Gentili E, Gerevini A, Gori M, Guns T, Homola M, Kersting K, Lehmann J, Lombardi M, Lorello L, Marconato E, Melacci S, Passerini A, Paul D, Riguzzi F, Teso S, Yorke-Smith N and Lippi M (2026) Benchmarking in neuro-symbolic ai. In: Dai WZ (ed.) *Learning and Reasoning*. Cham: Springer Nature Switzerland. ISBN 978-3-032-09087-4, pp. 238–249.

Manning C, Surdeanu M, Bauer J, Finkel J, Bethard S and McClosky D (2014) The Stanford CoreNLP natural language processing toolkit. In: Bontcheva K and Zhu J (eds.) *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Baltimore, Maryland: Association for Computational Linguistics, pp. 55–60. DOI:10.3115/v1/P14-5010. URL https://aclanthology.org/P14-5010/.

Marcus G (2020) The next decade in ai: Four steps towards robust artificial intelligence.

Marple K, Salazar E, Chen Z and Gupta G (2017) The s(asp) predicate answer set programming system. URL https://api.semanticscholar.org/CorpusID:195834851.

Marra G, Dumančić S, Manhaeve R and De Raedt L (2024) From statistical relational to neurosymbolic artificial intelligence: A survey. *Artificial Intelligence* 328: 104062. DOI:https://doi.org/10.1016/j.artint.2023.104062. URL https://www.sciencedirect.com/science/article/pii/S0004370223002084.

Mirzaee R, Rajaby Faghihi H, Ning Q and Kordjamshidi P (2021) SPARTQA: A textual question answering benchmark for spatial reasoning. In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online: Association for Computational Linguistics, pp. 4582–4598. DOI:10.18653/v1/2021.naacl-main.364. URL https://aclanthology.org/2021.naacl-main.364/.

Misino E, Marra G and Sansone E (2022) Vael: Bridging variational autoencoders and probabilistic logic programming. *ArXiv* abs/2202.04178. URL https://api.semanticscholar.org/CorpusID:246679982.

Mitra A and Baral C (2015) Learning to automatically solve logic grid puzzles. In: Màrquez L, Callison-Burch C and Su J (eds.) *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 1023–1033. DOI:10.18653/v1/D15-1118.

URL https://aclanthology.org/D15-1118/.

Muggleton S (1991) Inductive logic programming. *New Gen. Comput.* 8(4): 295–318. DOI:10.1007/BF03037089. URL https://doi.org/10.1007/BF03037089.

Möller M, Norlander A, Martires PZD and Raedt LD (2025) Neurosymbolic decision trees. URL https://arxiv.org/abs/2503.08762.

Nguyen QA, Pham TT, Vuong THY, Trinh VG and Thanh NH (2025) Detecting Misleading Information with LLMs and Explainable ASP. In: *ICAART 2025 - 17th International Conference on Agents and Artificial Intelligence*. Porto, Portugal: SCITEPRESS - Science and Technology Publications, pp. 1327–1334. DOI: 10.5220/0013357400003890. URL https://inria.hal.science/hal-04920600.

Odense S and d'Avila Garcez A (2025) A semantic framework for neurosymbolic computation. *Artificial Intelligence* 340: 104273. DOI:https://doi.org/10.1016/j.artint.2024.104273. URL https://www.sciencedirect.com/science/article/pii/S0004370224002091.

OpenAI (2023) Gpt-4 technical report.

Padalkar P, Wang H and Gupta G (2023) Using logic programming and kernel-grouping for improving interpretability of convolutional neural networks. In: Gebser M and Sergey I (eds.) *Practical Aspects of Declarative Languages*. Cham: Springer Nature Switzerland. ISBN 978-3-031-52038-9, pp. 134–150.

Padalkar P, Wang H and Gupta G (2024) Nesyfold: a framework for interpretable image classification. In: *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'24/IAAI'24/EAAI'24. AAAI Press. ISBN 978-1-57735-887-9, pp. 4378–4387. DOI:10.1609/aaai.v38i5.28235. URL https://doi.org/10.1609/aaai.v38i5.28235.

Padalkar P, Ślusarz N, Komendantskaya E and Gupta G (2025) A neurosymbolic framework for bias correction in convolutional neural networks. *Theory and Practice of Logic Programming* 24: 644–662. DOI:10.1017/S1471068424000322.

Parać R, Nodari L, Ardon L, Furelos-Blanco D, Cerutti F and Russo A (2024) Learning robust reward machines from noisy labels. In: *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning*, KR '24. ISBN 978-1-956792-05-8, pp. 909–919. DOI:10.24963/kr.2024/85. URL https://doi.org/10.24963/kr.2024/85.

Peng Y, Zha Z, Jin Y, Luo Z, Dai WZ, Ren Z, Ding YX and Zhou K (2025) Generating by understanding: Neural visual generation with logical symbol groundings. In: *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '25. New York, NY, USA: Association for Computing Machinery. ISBN 9798400714542, p. 2291–2302. DOI:10.1145/3711896.3736978. URL https://doi.org/10.1145/3711896.3736978.

Petersen F, Borgelt C, Kuehne H and Deussen O (2022) Deep differentiable logic gate networks. In: Oh AH, Agarwal A, Belgrave D and Cho K (eds.) *Advances in Neural Information Processing Systems*. pp. 2006–2018. URL https://openreview.net/forum?id=vF3WefcoePW.

Podell D, English Z, Lacey K, Blattmann A, Dockhorn T, Müller J, Penna J and Rombach R (2024) SDXL: Improving latent diffusion models for high-resolution image synthesis. *The Twelfth International Conference on Learning Representations* URL https://openreview.net/forum?id=di52zR8xgf.

Quattoni A and Torralba A (2009) Recognizing indoor scenes. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. pp. 413–420. DOI:10.1109/CVPR.2009.5206537.

Rader AP and Russo A (2023) Active learning in neurosymbolic ai with embed2sym. *COGAI@IJCLR* URL https://api.semanticscholar.org/CorpusID:269089085.

Ragno A, Plantevit M, Robardet C and Capobianco R (2024) Transparent explainable logic layers. In: *European Conference on Artificial Intelligence*. ISBN 9781643685489, pp. 914–921. DOI:10.3233/FAIA240579.

Rajasekharan A, Zeng Y and Gupta G (2023a) Argument analysis using answer set programming and semantics-guided large language models. *ICLP Workshops 2023* URL https://api.semanticscholar.org/CorpusID:259503415.

Rajasekharan A, Zeng Y, Padalkar P and Gupta G (2023b) Reliable natural language understanding with large language models and answer set programming. In: Pontelli E, Costantini S, Dodaro C, Gaggl S, Calegari R, D'Avila Garcez A, Fabiano F, Mileo A, Russo A and Toni F (eds.) Proceedings 39th International Conference on *Logic Programming,* Imperial College London, UK, 9th July 2023 - 15th July 2023, *Electronic Proceedings in Theoretical Computer Science*, volume 385. Open Publishing Association, pp. 274–287. DOI: 10.4204/EPTCS.385.27.

Redmon J and Farhadi A (2018) Yolov3: An incremental improvement. URL https://arxiv.org/abs/1804.02767.

Riegel R, Gray A, Luus F, Khan N, Makondo N, Akhalwaya IY, Qian H, Fagin R, Barahona F, Sharma U, Ikbal S, Karanam H, Neelam S, Likhyani A and Srivastava S (2020) Logical neural networks. URL https://arxiv.org/abs/2006.13155.

Riley H and Sridharan M (2019) Integrating non-monotonic logical reasoning and inductive learning with deep learning for explainable visual question answering. *Frontiers in Robotics and AI* Volume 6 - 2019. DOI:10.3389/frobt.2019.00125. URL https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2019.00125.

Ruis L, Andreas J, Baroni M, Bouchacourt D and Lake BM (2020) A benchmark for systematic generalization in grounded language understanding. In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20. Red Hook, NY, USA: Curran Associates Inc. ISBN 9781713829546, pp. 19861–19872.

Santana MB, Kareem I and Ricca F (2024) Towards automatic composition of asp programs from natural language specifications. In: *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, IJCAI '24. ISBN 978-1-956792-04-1, pp. 6198–6206. DOI:10.24963/ijcai.2024/685. URL https://doi.org/10.24963/ijcai.2024/685.

Sheth A, Roy K and Gaur M (2023) Neurosymbolic Artificial Intelligence (Why, What, and How) . *IEEE Intelligent Systems* 38(03): 56–62. DOI:10.1109/MIS.2023.3268724. URL https://doi.ieeecomputersociety.org/10.1109/MIS.2023.3268724.

Shi Z, Zhang Q and Lipani A (2021) Stepgame: A new benchmark for robust multi-hop spatial reasoning in texts. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36. pp. 11321–11329. DOI: 10.1609/aaai.v36i10.21383.

Singh D, Jain N, Jain P, Kayal P, Kumawat S and Batra N (2020) Plantdoc: A dataset for visual plant disease detection. In: *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD*, CoDS COMAD 2020. New York, NY, USA: Association for Computing Machinery. ISBN 9781450377386, p. 249–253. DOI:10.1145/3371158.3371196. URL https://doi.org/10.1145/3371158.3371196.

Sinha K, Sodhani S, Dong J, Pineau J and Hamilton WL (2019) CLUTRR: A diagnostic benchmark for inductive reasoning from text. In: Inui K, Jiang J, Ng V and Wan X (eds.) *Proceedings of the 2019 Conference on*

Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). Hong Kong, China: Association for Computational Linguistics, pp. 4506–4515. DOI:10.18653/v1/D19-1458. URL https://aclanthology.org/D19-1458/.

Skryagin A, Ochs D, Deibert P, Kohaut S, Dhami DS and Kersting K (2024a) Answer set networks: Casting answer set programming into deep learning. URL https://arxiv.org/abs/2412.14814.

Skryagin A, Ochs D, Dhami DS and Kersting K (2024b) Scalable neural-probabilistic answer set programming. *J. Artif. Int. Res.* 78. DOI:10.1613/jair.1.15027. URL https://doi.org/10.1613/jair.1.15027.

Skryagin A, Stammer W, Ochs D, Dhami DS and Kersting K (2022) Neural-Probabilistic Answer Set Programming. *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning* : 463–473DOI:10.24963/kr.2022/48. URL https://doi.org/10.24963/kr.2022/48.

Smet LD and Raedt LD (2025) Defining neurosymbolic ai. URL https://arxiv.org/abs/2507.11127.

Stallkamp J, Schlipsing M, Salmen J and Igel C (2012) Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks* 32: 323–332. DOI:https://doi.org/10.1016/j.neunet.2012.02.016. URL https://www.sciencedirect.com/science/article/pii/S0893608012000457. Selected Papers from IJCNN 2011.

Suchan J, Bhatt M and Varadarajan S (2021) Commonsense visual sensemaking for autonomous driving – on generalised neurosymbolic online abduction integrating vision and semantics. *Artificial Intelligence* 299: 103522. DOI:https://doi.org/10.1016/j.artint.2021.103522. URL https://www.sciencedirect.com/science/article/pii/S0004370221000734.

Tafjord O, Clark P, Gardner M, Yih Wt and Sabharwal A (2019) Quarel: a dataset and models for answering questions about qualitative relationships. In: *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press. ISBN 978-1-57735-809-1, pp. 7063–7071. DOI:10.1609/aaai.v33i01.33017063. URL https://doi.org/10.1609/aaai.v33i01.33017063.

Thomson A and Page D (2023) Neural markov prolog. URL https://arxiv.org/abs/2312.01521.

Tudor A and Gupta G (2024) Autonomous task completion based on goal-directed answer set programming. *ICLP Workshops* URL https://api.semanticscholar.org/CorpusID:276307594.

Wan Z, Liu CK, Yang H, Li C, You H, Fu Y, Wan C, Krishna T, Lin Y and Raychowdhury A (2024a) Towards cognitive ai systems: a survey and prospective on neuro-symbolic ai. URL https://arxiv.org/abs/2401.01040.

Wan Z, Liu CK, Yang H, Raj R, Li C, You H, Fu Y, Wan C, Samajdar A, Lin YC, Krishna T and Raychowdhury A (2024b) Towards cognitive ai systems: Workload and characterization of neuro-symbolic ai. In: *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. pp. 268–279. DOI:10.1109/ISPASS61541.2024.00033.

Wang H and Gupta G (2023) Fold-se: An efficient rule-based machine learning algorithm with scalable explainability. In: Gebser M and Sergey I (eds.) *Practical Aspects of Declarative Languages*. Cham: Springer Nature Switzerland, pp. 37–53.

Wang R, Sun K and Kuhn J (2024) Dspy-based neural-symbolic pipeline to enhance spatial reasoning in llms. URL https://arxiv.org/abs/2411.18564.

Weston J, Bordes A, Chopra S, Rush AM, van Merriënboer B, Joulin A and Mikolov T (2015) Towards ai-complete question answering: A set of prerequisite toy tasks. URL https://arxiv.org/abs/1502.05698.

Winters T, Marra G, Manhaeve R and Raedt LD (2022) Deepstochlog: Neural stochastic logic programming. *Proceedings of the AAAI Conference on Artificial Intelligence* 36(9): 10090–10100. DOI:10.1609/aaai.v36i9.21248. URL https://ojs.aaai.org/index.php/AAAI/article/view/21248.

Yang Z, Ishay A and Lee J (2020) Neurasp: Embracing neural networks into answer set programming. In: Bessiere C (ed.) *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. International Joint Conferences on Artificial Intelligence Organization, pp. 1755–1762. DOI: 10.24963/ijcai.2020/243. URL https://doi.org/10.24963/ijcai.2020/243. Main track.

Yang Z, Ishay A and Lee J (2023) Coupling large language models with logic programming for robust and general reasoning from text. In: *Findings of the Association for Computational Linguistics, ACL 2023*, Proceedings of the Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics (ACL), pp. 5186–5219. DOI:10.18653/v1/2023.findings-acl.321. Publisher Copyright: © 2023 Association for Computational Linguistics.; 61st Annual Meeting of the Association for Computational Linguistics, ACL 2023 ; Conference date: 09-07-2023 Through 14-07-2023.

Yu D, Yang B, Liu D, Wang H and Pan S (2023) A survey on neural-symbolic learning systems. *Neural Networks* 166: 105–126. DOI:https://doi.org/10.1016/j.neunet.2023.06.028. URL https://www.sciencedirect.com/science/article/pii/S0893608023003398.

Zeng Y, RAJASEKHARAN A, BASU K, WANG H, ARIAS J and GUPTA G (2024) A reliable common-sense reasoning socialbot built using llms and goal-directed asp. *Theory and Practice of Logic Programming* 24(4): 606–627. DOI:10.1017/s147106842400022x. URL http://dx.doi.org/10.1017/S147106842400022X.

Zeng Y, Rajasekharan A, Padalkar P, Basu K, Arias J and Gupta G (2023) Automated interactive domain-specific conversational agents that understand human dialogs. In: Gebser M and Sergey I (eds.) *Practical Aspects of Declarative Languages*. Cham: Springer Nature Switzerland. ISBN 978-3-031-52038-9, pp. 204–222.

Zhang H, Kung PN, Yoshida M, den Broeck GV and Peng N (2024) Adaptable logical control for large language models. URL https://arxiv.org/abs/2406.13892.

Zhou B, Lapedriza A, Khosla A, Oliva A and Torralba A (2018) Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40(6): 1452–1464. DOI: 10.1109/TPAMI.2017.2723009.