Journal Title
XX(X):1–33

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

A Survey of Neurosymbolic Answer Set Programming

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 both 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. 2023), 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 knowledge from examples. However, they lack explicit reasoning and logic capabilities, which can lead

[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

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 formally defining ASP in the background section, explaining how to learn ASP rules and providing a quick overview of neural networks. The main section is then divided into two parts.

The first part contains a discussion of papers in the field of neurosymbolic ASP. It is split into four sections, 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. We conclude that the advantages of neurosymbolic ASP lie in explainability, robustness and data efficiency, compared to purely neural approaches.

The second part discusses the current limits and open challenges of the field. It is split into three sections, covering simple perception tasks, limited ASP generation and scalability issues. We show that neurosymbolic ASP has so far been restricted to simple inputs that do not reflect real-world scenarios. Moreover, most frameworks rely on hand-written background knowledge to complement the knowledge that is learned, or 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.

This survey focuses on papers released within the last five years, i.e. from 2020 onwards. We only discuss frameworks that use an expressive subset of ASP, at least at the level of stratified programs. This excludes frameworks that learn simpler ASP rules, as well as those with sufficiently expressive rules in other logical languages. Finally, we restrict our focus to supervised learning and do not discuss work on combining reinforcement learning and ASP. Prominent papers outside our scope are discussed in the related work section.

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 $\mathtt{p}$. A **literal** is an atom $\mathtt{p}(\mathtt{t}_1, \ldots, \mathtt{t}_n)$ or its negation $\neg\mathtt{p}(\mathtt{t}_1, \ldots, \mathtt{t}_n)$. Terms without variables are called **ground** and an atom is ground if every term in it is ground. Ground atoms and their negations are ground literals. A set of ground literals is called an interpretation.

A general **rule** consists of literals $\mathtt{h}_1, \ldots, \mathtt{h}_k$ and $\mathtt{b}_1, \ldots, \mathtt{b}_n$ and has the form:

$$\mathtt{h}_1 \vee \cdots \vee \mathtt{h}_k :\!- \mathtt{b}_1, \ldots, \mathtt{b}_m, \mathtt{not}\ \mathtt{b}_{m+1}, \mathtt{not}\ \mathtt{b}_n$$

The left part of the rule is the **head** and the right part is the **body**. The head is a disjunction of literals 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 literals; the latter are indicated by the symbol `not` in front of them. Unlike classical negation, denoted by the symbol $\neg$, the symbol `not` denotes **negation as failure** and means that a literal is not believed to be true. If there are no negative literals in the body, it is a **definite** rule. A **program** is a collection of rules. A definite program consists of only definite rules and a stratified program contains no cycles through negation.

Semantics. A set S of ground literals satisfies

1. literal $\mathtt{l}$ if $\mathtt{l} \in S$,
2. $\mathtt{not}\ \mathtt{l}$ if $\mathtt{l} \notin S$,
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 literal 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 literals, 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 $\mathtt{not}\ \mathtt{l}$ such that $\mathtt{l} \in S$. Last, remove all remaining body literals containing `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

$$\mathtt{l}\{\mathtt{h}_1, \ldots, \mathtt{h}_k\}\mathtt{u} :\!- \mathtt{b}_1, \ldots, \mathtt{b}_m, \mathtt{not}\ \mathtt{b}_{m+1}, \ldots, \mathtt{not}\ \mathtt{b}_n$$

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. This construct is known as a **choice rule**.

Aggregates perform operations on sets. For example, the expression $\#\texttt{count}\{\texttt{X} : \texttt{p(X)}\}$ represents the number of elements in 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 $\#\texttt{maximize}\{\texttt{X} : \texttt{p(X)}\}$. 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 shorted 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. **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

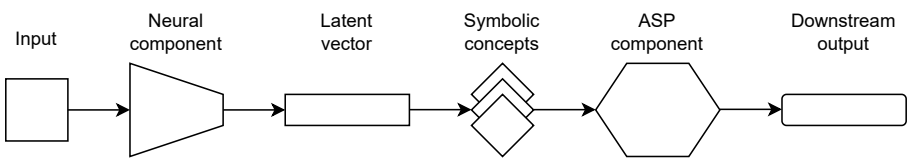


Figure 1. High-level depiction of inference through the components of neurosymbolic ASP frameworks.

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: $\texttt{result(Z)} :- \texttt{digit(1,X)}, \texttt{digit(2,Y)}, \texttt{Z} = \texttt{X} + \texttt{Y}$. The downstream output is the number `Z` in `result(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 a 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.

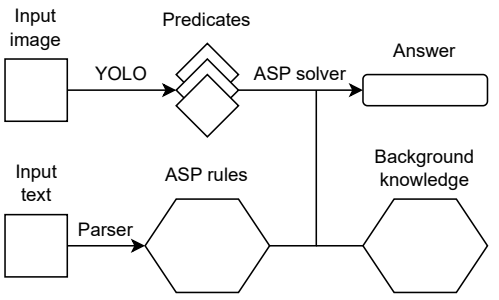
The proposed frameworks in the literature are all structured differently and deal with their own set of challenges. They can be split up into four broad categories:

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.

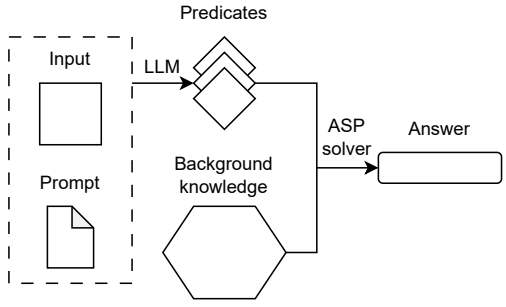
In this section, we discuss each category and illustrate the frameworks within it.

Pre-trained neural and hard-coded symbolic component

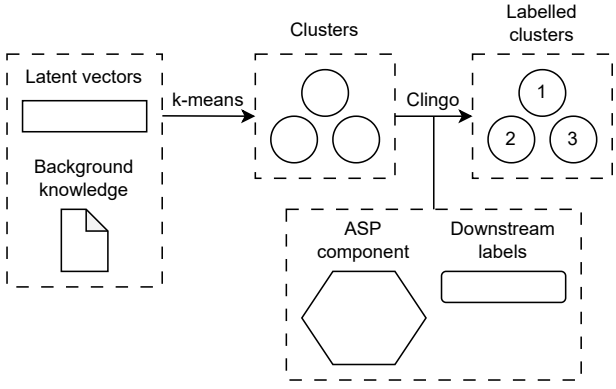
When both the neural and symbolic components are already given, the main focus lies on bridging the gap between them. Papers in this area tend to choose challenging tasks with natural language texts and complex images to demonstrate the usefulness of their framework. While earlier works use hand-crafted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



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



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



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

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

Figure 2a depicts the general structure of frameworks for the task 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 extracts 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.

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, ASP rules and background knowledge.

By adding a symbolic layer on top of the YOLO network, they 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.

Recently, LLMs have taken over the process of parsing natural language input. Their remarkable ability to produce structured language output from natural language input makes them well suited for the task of extracting ASP facts from text. They also require much less hand-crafting than building 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.

STAR. In a similar vein, the STAR framework (Rajasekharan et al. 2023) 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 using 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.

Embed2Sym. Unlike the previous frameworks, Embed2Sym (Aspis et al. 2022) pre-trains the neural component on downstream labels rather than latent 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 2c. 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, the frameworks in this section use symbolic components to improve the capabilities of neural networks and LLMs. They successfully apply parsers, LLM predictions or clustering to bridge the gap between raw data and ASP facts. Answers from such hybrid networks are more robust and explainable than those from the neural networks alone. However, the scope has been limited to the interplay between neural and ASP methods, where no neurosymbolic learning is involved. The lack of automatic learning procedures also leads to a need for labour-intensive hard-coding, which limits the 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.

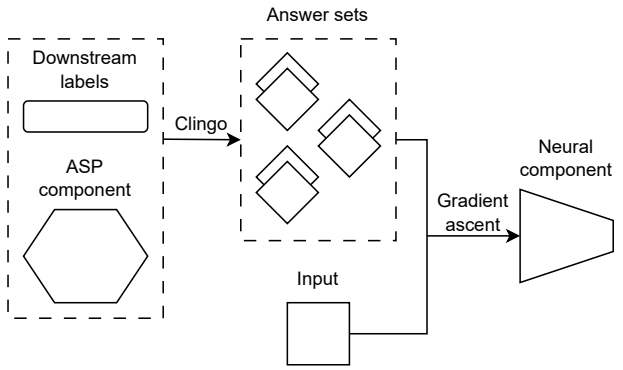


Figure 3. High-level depiction of the training procedure for the neural component in NeurASP (Yang et al. 2020), SLASH (Skryagin et al. 2024) and dPASP (Geh et al. 2024).

All frameworks in this section follow the same high-level procedure for training, which Figure 3 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. For each entry in the neural latent vector, the 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 noisy latent labels. Rather than having one label for a neural network prediction, you have a weighted set of 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. (2024), where the authors introduce a method called SAME to prune insignificant answer sets and speed up learning. As the neural predictions improve f 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.

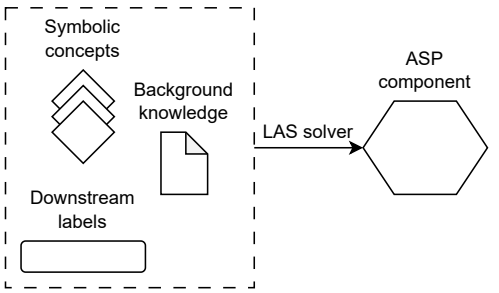
dPASP. Geh et al. (2024) introduce a more powerful specification language with dPASP. It extends the capabilities of NeurASP and SLASH by introducing interval-valued facts and disjunctions that are annotated with probabilities. They implement two semantics for their framework: maxent and credal. The 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.

Overall, the fundamental way of propagating the learning signal remains the same in all of the frameworks in this section. The downstream label is converted into noisy latent labels through calculating answer sets and assigning probabilities to them. This approach enables solving complex problems, while keeping the neural components simple. NeurASP introduced this notion and has been expanded upon with SLASH and dPASP. These models add capabilities like more complex probabilistic modelling and handling interval-valued or missing data.

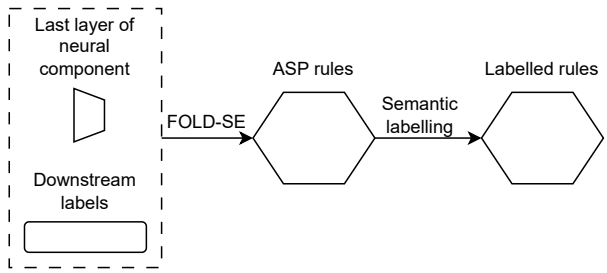
Symbolic learning with pre-trained neural component

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

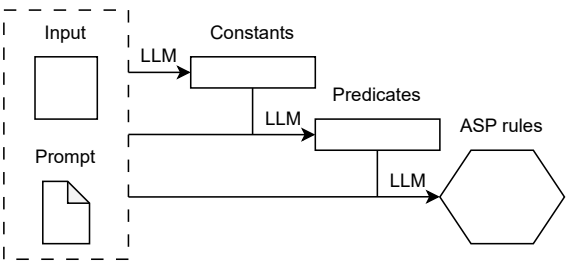
is large and neural outputs are noisy. Generating rules with LLMs is challenging as well, because the free-form output has to be syntactically and semantically correct. In this section, we discuss the different strategies that have been proposed to overcome these issues.



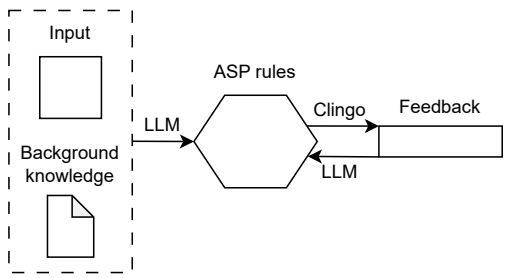
(a) High-level depiction of the symbolic learning procedure in FFNSL (Cunnington et al. 2023a), NeSyGPT (Cunnington et al. 2024) and Embed2Rule (Aspis et al. 2024).



(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 GPT-ASP (Ishay et al. 2023).



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

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

Figure 4a illustrates frameworks that deploy an off-the-shelf LAS solver like ILASP or FastLAS to find ASP rules. They differ in how they create the symbolic concepts necessary 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.

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 2c. 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.

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 three 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 the training sets include much less ASP than Python, LLMs struggle to generate correct answer set programs from scratch. Therefore, all approaches use multi-step methods.

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 4c 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 4d.

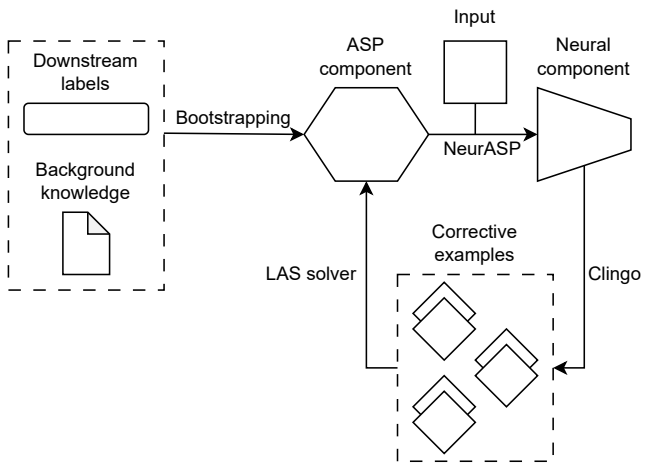
DSPy-ASP. In the DSPy-ASP framework (Wang et al. 2024), the LLM can revise the ASP rules for three iterations. First, it generates ASP predicates and queries, which are input to the Clingo solver together with predefined knowledge modules. Second, the LLM then revises the answer set program based on 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.

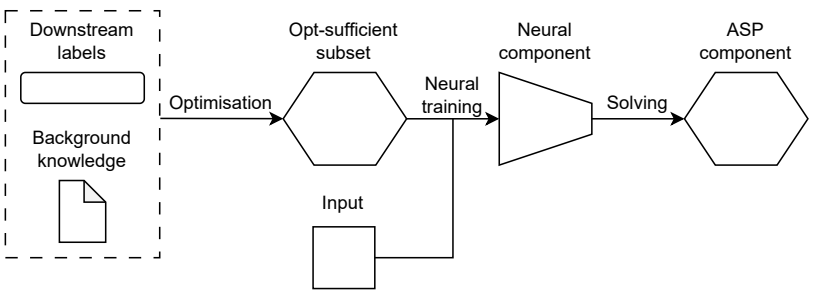
To sum up, there are two main approaches to learning ASP rules from raw data: Converting the data into symbolic examples to use with off-the-shelf solvers or generating rules directly. Both strands have been influenced by the rise of foundation models. In the former, VLMs act as the perception component, 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 5. 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 5a 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 5b. 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.

Performance analysis

The experimental sections of the literature provide an insight into the viability of the proposed methods on different datasets. They also compare their methods to purely neural or symbolic baselines, as well as other neurosymbolic frameworks. Overall, most experiments yield superior results in accuracy compared to baselines.

With regard to purely neural methods, authors also highlight advantages in robustness and explainability. FFNSL, for example, maintains a better performance under distributional shifts than a neural network alone. The STAR framework can generate justifications for each answer in form of

a proof tree. And while NesyFOLD's rules create a slight drop in performance compared to a CNN, they provide the ability to explain decisions and correct biases. Papers that combine LLMs and ASP report better results than using LLMs by themselves, especially on logically challenging tasks. Whether the advantage in reasoning has remained is hard to gauge, since LLMs are constantly improving their reasoning capabilities.

Compared to other logic-based methods, such as propositional logic or Prolog, ASP frameworks are more expressive. The ability of ASP to represent non-monotonic formulas allows it to solve problems that other logical frameworks fail at. For example, the NSIL paper includes experiments on the hitting set problem, which is NP-complete. NSIL manages to find a solution, while other baselines fail, because they can only express definite rules.

When papers compare their approach to other neurosymbolic ASP frameworks, they often highlight improvements in runtime speed. SLASH improves upon NeurASP by speeding up learning significantly through their SAME method, and dPASP runs quicker than NeurASP as well. Similarly, Embed2Sym can solve tasks where SLASH times out, as it uses clusters rather than having to compute answer sets. The same is the case for Embed2Rule compared to NSIL. NeuralFastLAS is faster than NSIL in tasks like 3-number-additions, but is unable to complete some problems, e.g. those that require predicate invention.

Papers report improvements in explainability, robustness and often accuracy when adding ASP to neural methods. However, there are certain limitations in these experiments. The tasks are often easy, the baselines simple and much of the necessary knowledge is hard-coded. In the next section, we will take a closer look at the drawbacks of neurosymbolic ASP and define the limits of current developments in the field.

The limits of neurosymbolic ASP

There are overarching limits in this research area that current methods have not been able to overcome and hold the field back. In this section, we aim to identify where these limits lie and what is needed to break through them. We broadly categorise them into three themes: simple perception components, a limited capacity to generate ASP and scalability issues.

Simple perception tasks

The datasets for the vast majority of papers include very simple perception tasks. This, in turn, means that the perception components tend to be small neural networks that have limited capabilities. While the frameworks might be able to train such small networks, it puts into question their usability in real-world scenarios.

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.

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. Out of the twelve frameworks in this survey that take images as inputs, nine of them are tested on MNIST images, despite its simplicity. The main reason is that the image classification only forms the

Synthetic images	Real-world images	Natural language text
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	**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), **Indoor scenes** (Quattoni and Torralba 2009): FFNSL **PlantDoc** (Singh et al. 2020): NeSyGPT **Places** (Zhou et al. 2018), **German traffic signs** (Stallkamp et al. 2012): NeSyFOLD	**bAbI** (Weston et al. 2015), **CLUTRR** (Sinha et al. 2019), **gSCAN** (Ruis et al. 2020): [LLM]+ASP **StepGame** (Shi et al. 2021): [LLM]+ASP, DSPy-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

Table 1. Datasets used for the perception components and the frameworks that use them.

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 `has_attributes(object1,red,circle,bright,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.

In the category of synthetic images, only MNIST is truly used for neurosymbolic training of the perception component, where the downstream label only provides a noisy signal to the latent classification task. Both ShapeWorld and CLEVR, while embodying a more complex perception task, provide direct latent labels for the frameworks.

Real-world images. CIFAR-10 is a dataset of real-world objects, but the images are compressed to 32x32 pixels and include only 10 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 marginally 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 issue 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, such as "Follow Suit". In that game, the winner is the player with the same suit as player 1 and the highest rank. The downstream label only provides the number of the winning player, not the ranks or suits of the cards. The Follow Suit task is therefore a dataset comprising both real-world inputs and downstream labels that do not provide strong latent signals. However, 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. The 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). 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.

StepGame contains questions that require multi-hop spatial reasoning of up to 10 steps. 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. 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. The authors have also created multiple chatbots based on this framework and have done qualitative testing using real user input. They conclude that STAR performs better than vanilla LLMs in metrics such as staying on topic or providing relevant responses.

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, no training of the neural component is occurring.

Limited 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 and AQuA utilise YOLO, while [LLM]+ASP and STAR 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.

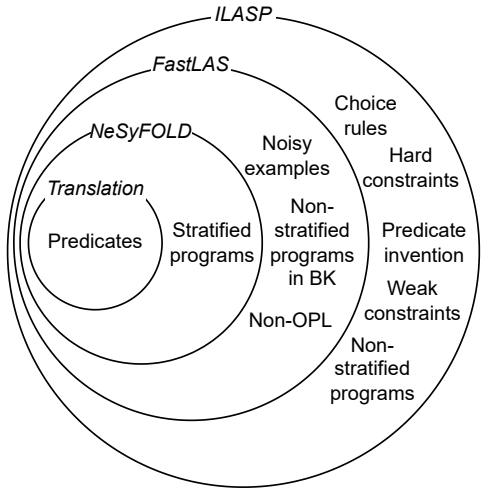


Figure 6. Levels of expressivity of different LAS frameworks.

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. They all support the use of ILASP as well for tasks that require higher expressive power. NSIL uses ILASP's capability to learn choice rules in the Hitting set task, where the framework needs to generate all hitting sets of a given collection. FFNSL, NeSyGPT and Embed2Rule require its predicate invention capability for the Follow suit task. Figure 6 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 two frameworks that use LLMs to produce entire ASP rulesets: GPT-ASP generates rules in a four step process and LLM-ARC generates both ASP rules and tests, refining them iteratively. 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.

In summary, the expressive capabilities of generated answer set programs vary between frameworks. Four of them use ILASP for generation, which gives them the ability to generate complex programs, including constructs such as negation as failure. However, they need to help ILASP out significantly by utilising background knowledge and mode biases to restrict the search space, which we discuss in the next section. Only two LLM-based frameworks generate rules directly, rather than just extracting predicates. They can theoretically create rules of any level of expressivity. But in practice they have only generated deductive steps for solving logic puzzles, rather than learning inductive knowledge like ILASP does. We explore this further in the Induction vs Deduction section.

Background knowledge. Out of the 17 frameworks discussed in this survey, nine hard-code the ASP component. Out of the remaining frameworks, five use FastLAS or ILASP and therefore make extensive use of background knowledge. FFNSL, NeSyGPT, Embed2Rule, NSIL and NeuralFastLAS all 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.

Only three frameworks generate 100% of their ASP rules themselves: NeSyFOLD, GPT-ASP and LLM-ARC. NeSyFOLD restricts the search space by only considering stratified ASP rules. The latter two papers use LLMs and guide the search through in-context learning, where a few solved examples are put in the prompt. However, they solve a fundamentally different task than the traditional models: They create a new program for each question, rather than coming up with a general solution for a task from a set of examples. This discrepancy is discussed in the next section.

Induction vs deduction. Learning answer set programs has traditionally been done 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 issues

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 limits the scalability of frameworks that make use of ILASP, such as FFNSL, NeSyGPT or Embed2Rule. 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. However, they themselves require a lot of resources. All LLM-based frameworks in this survey were run on a version of OpenAI's GPT models, either GPT-3 or GPT-4. These are proprietary models that require dedicated server architecture to run and usually incur 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.

Training traditional neural networks in a neurosymbolic way comes with scalability issues too. NeurASP-based methods, which include SLASH, dPASP and NSIL, calculate all answer sets for the given answer set program when training the neural network. This is feasible for tasks like MNIST Addition, where there are at most 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. Aspis et al. (2024) report that SLASH and NSIL are timing out or running out of memory in Follow suit and Sudoku.

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, ASP methods like ILASP or NeurASP run on the CPU. We need faster implementations, in addition to theoretical discoveries, to make neurosymbolic ASP methods more viable in real-world tasks.

Related work

In this survey, we have discussed papers that combine neural networks and ASP. We restricted our focus to supervised machine learning and required that the frameworks use rules at least as expressive as stratified answer set programs. In this section, we will briefly discuss papers related to neurosymbolic ASP that do not fulfill all these criteria.

Other logical languages

Many papers have proposed neurosymbolic algorithms in logical languages other than ASP. There are too many to name, so we will briefly discuss the most well-known.

DeepProbLog (Manhaeve et al. 2021) is an extension of the logical language ProbLog, which integrates neural networks for specifying probabilities of facts. Belonging to the family of Prolog, it is capable of expressing stratified programs.

MetaABD (Dai and Muggleton 2021) trains a neural network and induces a logical theory jointly. The framework uses a combination of abduction and induction and can learn recursive first-order theories with predicate invention. The rules are limited to definite logic, which only accept one model, rather than the many-world semantics in ASP. This means that it cannot model constructs such as defaults, exceptions, constraints or choices.

Evans et al. (2021) tackle a slightly different problem with the Apperception task. The goal is to build a theory that predicts future states given temporal data. The language of the theories is in a temporal Datalog variation, but the binary neural network to map raw data to symbols is encoded in ASP.

Subsets of ASP

Some papers use ASP as their logical language, but limit themselves to a subset of it. One example is Pix2Rule, which extracts rules from a neural layer and represents them in ASP (Cingillioglu and Russo 2021). However, the rules are in disjunctive normal form (DNF), which is propositional logic. The DNF-EO framework expands Pix2Rule to real-world multi-classification tasks, but still represents all extracted rules in DNF (Baugh et al. 2023).

Riley and Sridharan (2019) propose a framework that uses non-monotonic reasoning in CR-Prolog, which is a subset of ASP. They use hardcoded rules to learn classifications given features extracted by CNNs. They also include a mechanism to learn rules, which trains a decision tree and translates it into logic. This restricts the expressiveness of learned rules, as they do not use constructs like negation as failure.

Reinforcement learning

ASP is also used as a specification language for rewards in reinforcement learning (RL). A variety of papers specify goals in ASP and some even learn such rules.

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 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 Clingo. 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.

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.

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. They achieve good results on complex tasks by using either elaborate pipelines or LLMs for the translation into ASP. When the neural components need to be trained, the main challenge lies in propagating the learning signal through the non-differentiable ASP component. All papers in this category essentially use the same technique: enumerating answer sets to serve as noisy labels. This leaves room for further research into different methods of providing learning signals. When the neural network is pre-trained, its predictions serve as noisy examples for learning ASP rules. The challenge here lies with the limitations of rule learners. Traditional methods like ILASP require hard-coding rule templates to restrict the search space. Although LLMs can translate natural language problems into ASP rules, it has not yet been demonstrated that they can learn general rules inductively. 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 they are limited to simple perception problems.

The main open challenges in neurosymbolic ASP include harder tasks, less hard-coding and better scalability. The current limit for joint learning in perception tasks is CIFAR-10, while the majority of language datasets use synthetically generated sentences. More realistic tasks are needed to demonstrate the applicability of neurosymbolic ASP in real-world settings. The need for extensive hard-coding also holds the field back. Pre-training the neural component, especially foundation models, and hand-writing rules can be a very expensive processes. Even frameworks that learn rules need to limit the search space with manually engineered rule templates. Lastly, symbolic methods lack scalability compared to neural frameworks like Pytorch or Tensorflow. 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.

To overcome these challenges, solutions are needed in multiple areas. More efficient methods are necessary for scaling up to larger tasks. This includes improving the implementation and speed of existing methods, such as Clingo, ILASP or NeurASP, 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 of the inefficiencies. Foundation models also represent a promising direction, if enough resources are available to use and finetune them. Further research is necessary to learn inductive knowledge with LLMs, for example by integrating them with LAS solvers.

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 mix of expressiveness and readability, ASP is well-suited to fulfill this role, making neurosymbolic ASP a worthwhile area for further research.

Acknowledgements

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

References

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.

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.

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.

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.

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

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.

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.

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

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.

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.

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

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.

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/.

Hua W, Sun F, Pan L, Jardine A and Wang WY (2025) Inductionbench: LLMs fail in the simplest complexity class. In: *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, 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.

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.

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.

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.

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.

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.

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 (2023) Summary of chatgpt/gpt-4 research and perspective towards the future of large language models.

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.

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.

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/.

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.

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.

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.

Rajasekharan A, Zeng Y, Padalkar P and Gupta G (2023) 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.

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.

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, Dhami DS and Kersting K (2024) 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.

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.

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. DOI:10.1609/aaai.v33i01.33017063. URL https://doi.org/10.1609/aaai.v33i01.33017063.

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.

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.

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.

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.

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.