TPLP: Page 1–25.

doi:10.1017/S1471068425100343

Question Answering with LLMs and Learning from Answer Sets

MANUEL ALEJANDRO BORROTO SANTANA
Department of Mathematics and Computer Science, University of Calabria, Arcavacata, Italy
(*e-mail:* manuel.borroto@unical.it)

KATIE GALLAGHER
The University of Chicago, Chicago, IL, USA
(*e-mail:* krgallagher@uchicago.edu)

ANTONIO IELO, IRFAN KAREEM and FRANCESCO RICCA
Department of Mathematics and Computer Science, University of Calabria, Arcavacata, Italy
(*e-mails:* antonio.ielo@unical.it, irfan.kareem@unical.it, francesco.ricca@unical.it)

ALESSANDRA RUSSO
Department of Computing, Imperial College London, London, UK
(*e-mail:* a.russo@imperial.ac.uk)

submitted 24 March 2025; revised 28 August 2025; accepted 30 September 2025

Abstract

Large language models (LLMs) excel at understanding natural language but struggle with explicit commonsense reasoning. A recent trend of research suggests that the combination of LLM with robust symbolic reasoning systems can overcome this problem on story-based question answering (Q&A) tasks. In this setting, existing approaches typically depend on human expertise to manually craft the symbolic component. We argue, however, that this component can also be automatically learned from examples. In this work, we introduce LLM2LAS, a hybrid system that effectively combines the natural language understanding capabilities of LLMs, the rule induction power of the learning from answer sets (LAS) system ILASP, and the formal reasoning strengths of answer set programming (ASP). LLMs are used to extract semantic structures from text, which ILASP then transforms into interpretable logic rules. These rules allow an ASP solver to perform precise and consistent reasoning, enabling correct answers to previously unseen questions. Empirical results outline the strengths and weaknesses of our automatic approach for learning and reasoning in a story-based Q&A benchmark.

KEYWORDS: logic-based learning knowledge representation, question and answering (Q&A)

1 Introduction

One of the longstanding challenges in artificial intelligence (AI) is equipping machines with the ability to perform commonsense reasoning and to learn such knowledge autonomously from experience or text (Davis and Marcus 2015). This involves not only understanding implicit knowledge about the world but also applying it flexibly to novel situations, an ability that remains difficult for current AI systems. Nonetheless, in machine comprehension and question answering (Q&A) tasks, AI models frequently achieve high performance by exploiting statistical regularities and shallow text patterns rather than through the acquisition of explicit commonsense knowledge or the execution of systematic reasoning processes (Al-Negheimish *et al*. 2021).

Despite their impressive recent successes, LLMs are no exception to the broader limitations of current AI systems. They have been shown to exhibit limited reasoning capabilities and often generate unfaithful or incorrect answers (Zheng *et al*. 2023), leading to underperformance on benchmarks specifically designed to evaluate natural language reasoning (Lake and Murphy 2020; Wei *et al.* 2022). While recent techniques, such as chain-of-thought (CoT) prompting (Wei *et al*. 2022), problem decomposition, and in-context learning (Zhao *et al*. 2023), suggest that these models can exhibit some reasoning-like behavior, their capabilities remain limited, often relying on implicit pattern matching rather than robust, generalizable reasoning mechanisms (Lake and Murphy 2020; Wei *et al*. 2022). Moreover, the lack of transparency and explainability in LLMs makes it challenging to determine whether they truly acquire and apply commonsense reasoning (Sap *et al*. 2020).

On the other hand, LLMs have demonstrated strong capabilities in processing and generating natural language text. Notably, they have proven effective in semantic parsing, the task of translating natural language sentences into formal representations (Drozdov *et al.* 2023). This ability positions LLMs as valuable components for bridging the gap between unstructured language and structured, machine-interpretable logic. Recent neuro-symbolic approaches integrate LLMs into formal reasoning frameworks (Kautz 2022), exploiting their effectiveness in translating natural language into structured representation. This line of research demonstrates that such combinations can address some of the inherent limitations of LLMs, particularly their lack of explicit reasoning and factual reliability, while retaining their strengths in language generation and semantic interpretation. For instance, it has been shown that the coherence and consistency of LLMs in story completion tasks can be significantly enhanced by combining LLM-based semantic parsing (to translate text into formal representations) with symbolic reasoning systems that evaluate the correctness of the LLM-generated sentences (Nye *et al.* 2021). Moreover, Ishay *et al*., combine LLMs with answer set programming (ASP) (Lifschitz 2008; Brewka *et al*. 2011) to solve logic puzzles (Ishay *et al*. 2023). Yang *et al*., combine GPT-3-based semantic parsing with an ASP knowledge module to perform reasoning, showing state-of-the-art performance on several benchmarks (Yang *et al.* 2023). These approaches also demonstrated that ASP, due to its expressive and robust declarative semantics, is a particularly well-suited symbolic formalism for supporting reasoning in neuro-symbolic systems. However, despite yielding more robust and

https://doi.org/10.1017/S1471068425100343 Published online by Cambridge University Press

interpretable reasoning over textual inputs, they typically depend on manually crafted symbolic knowledge for the reasoning component. This manual intervention is time consuming, requires substantial domain expertise, and results in an explicit limitation to scalability and generalization across diverse tasks or domains.

We claim that the symbolic component need not be manually specified, but can instead be automatically learned from examples. Through our proposed approach, the feasibility of this direction is demonstrated, showing that meaningful and generalizable symbolic knowledge can be induced from limited supervision. More in detail, we develop the ideas of combining ASP with LLMs (Ishay *et al*. 2023) for robust reasoning, and introduce LLM2LAS, a hybrid system that effectively combines the natural language understanding capabilities of LLMs, the rule induction power of the learning from answer sets (LAS) system ILASP (Law *et al*. 2020), and the formal reasoning strengths of ASP (Brewka *et al*. 2011).

LLM2LAS integrates an LLM-based semantic parser with ILASP, a system for inductive learning of knowledge in ASP specifications. The semantic parser extracts symbolic representations from natural language stories and questions, which are then used to automatically construct ILASP learning tasks. Given a story, along with associated questions and answers, LLM2LAS iteratively learns from narratives the underlying commonsense logic rules required to solve the task. The induced knowledge is general and transferable, enabling an ASP system to correctly answer questions about previously unseen texts.

The key components of LLM2LAS include:

- An open-source LLM-based few-shot semantic parser for generating from natural language both

  (i) ASP representations of the input stories, and
  (ii) mode bias declarations to drive LAS systems[1]

- A learning module built upon ILASP, designed to induce commonsense knowledge required for answering questions about narrative texts.
- A reasoning module for answering questions about a story using the learned commonsense knowledge, which is based on the `clingo` ASP solver (Gebser *et al*. 2019).

We evaluated our approach on the bAbI Q&A dataset (Weston *et al*. 2016), a widely used benchmark comprising several tasks designed to test various forms of reasoning, including deduction, induction, coreference resolution, and temporal reasoning. The empirical evaluation highlights both the strengths and current limitations of our automated approach to learning and reasoning in story-based Q&A tasks. LLM2LAS represents a promising step toward the development of more autonomous, interpretable, and robust systems capable of reasoning over natural language inputs.

[1] Mode bias (Law *et al*. 2020) is a form of syntactic constraint that defines the set of logic rules that the system is allowed to consider when learning, see Section 3.3.

https://doi.org/10.1017/S1471068425100343 Published online by Cambridge University Press

2 Related work

Mitra *et al.*, developed a three-layer Q&A system that combines statistical methods with inductive rule learning and reasoning (Mitra and Baral 2016). The system includes a statistical inference layer, which uses an abstract meaning representation (AMR) parser, a translation layer, which converts the AMR parser output into Event Calculus (EC) syntax using a naive deterministic algorithm, and the reasoner layer, which uses a modified version of the inductive logic programming (ILP) Cropper and Dumancic 2022) algorithm XHAIL (Ray 2009) to learn the knowledge required for reasoning. The system achieves on the bAbI dataset an accuracy of 99.68%, but requires users to manually specify mode declarations and task-dependent background knowledge.

Nye *et al.*, proposed a neuro-symbolic approach to improve the coherence and consistency of text generation in a story completion task (Nye *et al.* 2021). The approach uses GPT-3 to generate candidate completion sentences and an LLM-based parser to derive logical representations of a given story and generated sentences. The latter are compared to symbolic candidates inferred using a minimal world model to check consistency. Only consistent candidates are considered for the final generation. The system performs well on different benchmarks (Weston *et al.* 2016; Sinha *et al.* 2019; Ruis *et al.* 2020). However, the main limitation is the manual design of the world model, which is task specific.

Ishay *et al.*, combined LLM and ASP to solve logical puzzles in a step-by-step manner (Ishay *et al.* 2023). The method uses GPT-3 with prompt engineering to extract relevant objects, their categories and typed predicates from text descriptions of the puzzles. It then generates an ASP program that captures the rules of the given puzzle, using a Generate-Define-Test approach. The outcomes are computed symbolically using the generated ASP program. The method is interpretable but requires human intervention to resolve errors in the generation process.

Yang *et al.*, demonstrated GPT-3 to be effective in few-shot semantic parsing of natural language into ASP representation (Yang *et al.* 2023). Their approach handles Q&A tasks but with task-specific manually handcrafted background knowledge, achieving promising results on different NLP benchmarks, included the bAbI dataset. The authors also conducted additional experiments to evaluate the capacity of LLMs themselves to handle reasoning tasks. They used a generation-only approach based on GPT-3.5 and various prompting techniques (i.e., Few-shot and CoT). The results demonstrated that, while LLMs can achieve decent results on some tasks, their overall performance is significantly lower compared to the proposed neuro-symbolic approach. Our approach differs from this work in that we *learn the relevant knowledge needed to solve a task*.

Alviano *et al.* in their first (Alviano and Grillo 2024) and second report (Alviano *et al.* 2024), introduced the LLM2ASP framework, which integrates the reasoning capabilities of ASP with the natural language processing capabilities of LLMs. They proposed a YAML-based format for specifying prompts and encoding domain-specific background knowledge. In this framework, LLMs process the input prompt to generate relational facts or ground truth, which are reasoned upon using an ASP program. The resulting output from the ASP program is converted back into natural language using LLMs to provide a better user experience. Kim *et al.* (2024) addressed the reasoning capabilities of

https://doi.org/10.1017/S1471068425100343 Published online by Cambridge University Press

black-box LLMs. They proposed a novel approach called correct for improving QA reasoning of black-box (COBB) LLMs. The approach utilizes a trained adaptation model to map the often-imperfect reasoning of an initial black-box LLM to the correct reasoning. The adaptation model is based on an open-source LLM model and trained over a set of representative pairs of correct and incorrect reasoning. The proposed approach's effectiveness depends on the quality of training pairs and the capability of open-source LLM. In addition, it requires ground-truth human labels to judge the correctness of reasoning, which is a time-consuming task.

Wu *et al*. (2024) proposed MindMap, a fully LLM-based approach to enhance the multi-step reasoning in LLMs by constructing evidence chains of facts associated with a common subject. The approach puts the related facts together to prevent missing crucial information. The chains created by MindMap can be combined with CoT and selection-inference (SI) to improve the performance in logical reasoning tasks. The framework consists of three main modules, that is (i) evidence chain construction, (ii) chain summarization, and (iii) chain utilization for reasoning. The approach was evaluated on a subset of the bAbI dataset (tasks 1–3) and the ProofWriter (Tafjord *et al*. 2021) dataset, demonstrating that integrating MindMap with CoT and SI leads to significant improvements. Despite these clear improvements, the overall performance remains below that of neuro-symbolic approaches, with hallucinations during inference representing a significant contributing factor. These results highlights that, despite recent progress, obtaining accurate and consistent reasoning from LLMs remains a challenge.

In addition to the approaches discussed above, there are other neuro-symbolic methods that address similar problems while relying on symbolic formalisms other than ASP, such as Prolog (Colmerauer and Roussel 1993) and constraint programming (Apt 2003). Recent surveys (Luo *et al*. 2023; Cheng *et al*. 2025) provide an up-do-date overview of these neuro-symbolic approaches.

We adopt LAS to learn the knowledge needed to solve a Q&A task, thus reducing human intervention, and exploit LLM-based semantic parsing capability to automatically generate LAS learning tasks from the given natural language dataset. This combination of LLM and LAS is novel and offers promising performance.

This paper is an extended and revised version of the conference paper by Kareem *et al*. (2024). In particular, this paper streamlines the ideas of Kareem *et al*., by adopting a simpler workflow, that replaces classic NLP techniques with LLM-based techniques, replacing parts-of-speech algorithms with few-shot prompting to define the learning bias for the tasks. Furthermore, several extensions were introduced in the implementation, ranging from more up-to-date LLMs (LLama 3.3 70B in place of the smaller model Falcon 7B) to smarter caching strategies for LLM outputs and learning tasks' hypothesis space. This extension makes the approach more flexible and expands its applicability. As a result, we are able to solve more tasks from the bAbI dataset, that were unfeasible in Kareem *et al*. (2024) due to the complexity of fact extraction.

3 Preliminaries

This section consists of a brief recap on ASP (Section 3.1), the EC formalism to reason about actions (Section 3.2), the inductive logic programming under the LAS framework

https://doi.org/10.1017/S1471068425100343 Published online by Cambridge University Press

(Section 3.3), and large language models (LLMs) (Section 3.4), providing relevant notions that will be referred to throughout the paper.

3.1 Answer set programming

ASP is a well-known paradigm for specifying real-world problems, commonsense knowledge and solving combinatorial optimization problems (Gelfond and Lifschitz 1988; Brewka *et al.* 2011). We provide here a brief recap of the ASP syntax relevant to this paper, referring the reader to (Gelfond and Lifschitz 1988; Brewka *et al*. 2011; Calimeri *et al.* 2020) for a formal account on ASP syntax and semantics.

Syntax

Given atoms $\mathtt{h}$, $\mathtt{b}_1, \ldots, \mathtt{b}_n$, $\mathtt{c}_1, \ldots, \mathtt{c}_m$, a *normal rule* is of the form $\mathtt{h} :- \mathtt{b}_1, \ldots, \mathtt{b}_n, \mathtt{not}\ \mathtt{c}_1, \ldots, \mathtt{not}\ \mathtt{c}_m$, where $\mathtt{h}$ is the *head*, $\mathtt{b}_1, \ldots, \mathtt{b}_n$, $\mathtt{not}\ \mathtt{c}_1, \ldots, \mathtt{not}\ \mathtt{c}_m$ (collectively) is the *body* of the rule, and "not" represents negation as failure. Rules $:- \mathtt{b}_1, \ldots, \mathtt{b}_n$, $\mathtt{not}\ \mathtt{c}_1, \ldots, \mathtt{not}\ \mathtt{c}_m$ are called *hard constraints*. ASP programs include also *choice rules*. A choice rule is a special type of rule of the form $\mathtt{l}\{\mathtt{h}_1, \ldots, \mathtt{h}_k\}\mathtt{u} :- \mathtt{b}_1, \ldots, \mathtt{b}_n$, $\mathtt{not}\ \mathtt{c}_1, \ldots, \mathtt{not}\ \mathtt{c}_m$, where $\mathtt{l}$ and $\mathtt{u}$ are integers. A variable in a rule is said to be *safe* if it occurs in at least one positive literal (i.e., the $\mathtt{b}_i$'s in the above rule) in the body of the rule. In this paper, we assume an ASP program to be a set of normal rules, hard constraints, and choice rules. The semantics of ASP programs is in terms of stable models (or answer sets) (Gelfond and Lifschitz 1988).

ASP solvers are capable of constructing solutions to real-world problems from a given ASP program specification of the problem and, where needed, ranking solutions according to optimization criteria.

Semantics.

The Herbrand Base of a program P, denoted HB_P, is the set of variable free (ground) atoms that can be formed from predicates and constants in P. The subsets of HB_P are called the (Herbrand) interpretations of P. A ground aggregate $\mathtt{l}\{\mathtt{h}_1, \ldots, \mathtt{h}_k\}\mathtt{u}$ is satisfied by an interpretation I iff $\mathtt{l} \leq |I \cap \{\mathtt{h}_1, \ldots, \mathtt{h}_k\}| \leq \mathtt{u}$.

As we restrict our ASP programs to sets of normal rules, constraints, and choice rules, we can use the simplified definitions of the *reduct* for choice rules presented in Law *et al.* (2015). Given a program P and an Herbrand interpretation $I \subseteq HB_P$, the reduct P^I is constructed from the grounding of P in 4 steps. Firstly, removing rules whose bodies contain the negation of an atom in I; secondly, removing all negative literals from the remaining rules; thirdly, replacing the head of any constraint, or any choice rule whose head is not satisfied by I with $\bot$ (where $\bot \notin HB_P$); finally, replacing any remaining choice rule $\mathtt{l}\{\mathtt{h}_1, \ldots, \mathtt{h}_m\}\mathtt{u} :- \mathtt{b}_1, \ldots, \mathtt{b}_n$ with the set of rules $\{\mathtt{h}_i :- \mathtt{b}_1, \ldots, \mathtt{b}_n \mid \mathtt{h}_i \in I \cap \{\mathtt{h}_1, \ldots, \mathtt{h}_m\}\}$. Any $I \subseteq HB_P$ is an *answer set* of P if it is the minimal model of the *reduct* P^I. We denote with $AS(P)$ the set of answer sets of a program P. A program P is said to be satisfiable (resp. unsatisfiable) if $AS(P)$ is non-empty (resp. empty).

https://doi.org/10.1017/S1471068425100343 Published online by Cambridge University Press

Table 1. *Predicates to model Event Calculus as a normal logic program*

Predicate	Meaning
`holdsAt(f,t)`	The fluent f is true at t
`happensAt(f,t)`	The fluent f is observed at t
`initiatedAt(f,t)`	The fluent f initiates at t
`terminatedAt(f,t)`	The fluent f ceases at t

```
holdsAt(F,T+1):-initiatedAt(F,T),time(T).
holdsAt(F,T+1):-holdsAt(F,T),not terminatedAt(F,T),time(T).
```

Fig 1. Simple discrete event calculus axioms as ASP rules.

3.2 Simplified Discrete Event Calculus

EC (Kowalski and Sergot 1986) is a logic-based formalism to reason about actions and their effects. The EC formalization of a subject domain consists of a set of first-order rules that define properties of interest in the domain ("fluents"), and domain-independent rules ("axioms") that describe general principles about how such properties evolve, that is when, how they become true or false in a given point in time (Shanahan 1999). There exist multiple flavors of EC. In this paper, we are interested in the *simplified discrete event calculus (SDEC) (Katzouris et al. 2015a).*

SDEC can be elegantly implemented in ASP by means of a normal logic program, using predicates `holdsAt/2`, `initiatedAt/2`, and `terminatedAt/2`. Intuitively, SDEC consists of rules that enable to track (and infer) the truth value of fluents over a finite, discrete, linear representation of time.

The axioms of SDEC can be rendered in ASP according to the rules in Figure 1, and Table 1 reports the informal meaning of such predicates. The `initiatedAt/2` and `terminatedAt/2` predicates are used to define the point in times where an event initiates and terminates. Indeed, different fluents have different initiating and termination conditions. The predicate `holdsAt/2` tracks true fluents at any given time point, with `holdsAt(f,t)` modeling that fluent `f` is true at time t.

```
holdsAt(F,T+1):-initiatedAt(F,T),time(T).        (1)
```

```
holdsAt(F,T+1):-holdsAt(F,T),not terminatedAt(F,T),time(T).        (2)
```

3.2.1 Modeling narratives with event calculus

We provide an example of such ASP-based formalization of *narratives* by means of SDEC. A narrative is an ordered sequence of (natural language) statements that describes an event.

https://doi.org/10.1017/S1471068425100343 Published online by Cambridge University Press

Example 1.
Consider the following narrative, similar to those in Task 8 of the bAbI dataset. Each line consists of a sentence, and we assume that actions that take place in the i-th sentence happen at time i.

```
1. John is carrying the football.
2. John went to the kitchen.
3. John got an apple there.
4. John went to the park.
5. John got the baseball there.
6. John dropped the football.

   > What is John carrying?
```

The narrative involves the agent John, and its action involves interacting with items – picking them up, dropping them – as well as the moving through several locations.

The narrative provides explicit, point-wise, information about how John interacts with items and moves in space; however, the concept of "what is John carrying at any given point in time" is not explicitly provided in the narrative, but is implicit in *what it has been picked up, but not dropped yet*. The first step to model such a narrative in SDEC would be to appropriately choose fluents, and then to provide definitions for its initiating and terminating conditions.

In particular, a possible way to model such scenario is to use the fluent $\texttt{got}(\texttt{john}, \texttt{obj})$ to state that john picks up a given object, and $\texttt{drop}(\texttt{john}, \texttt{obj})$ to state he drops an object. Furthermore, the fluent $\texttt{carries}(\texttt{john}, \texttt{obj})$ states that John is *carrying* a specific item. Indeed, for completeness, one may also wish to include the fluent $\texttt{go\_to}(\texttt{john}, \texttt{loc})$ to state that John is moving to a specific location `loc`, however notice that in this particular case, it is not necessary to keep track of John's location to answer the narrative's question. Thus, we can *reify* the narrative by means of the following ASP facts:

$$\texttt{initiatedAt}(\texttt{carry}(\texttt{john},\texttt{football}),1).$$

$$\texttt{happensAt}(\texttt{go\_to}(\texttt{john},\texttt{kitchen}),2).$$

$$\texttt{happensAt}(\texttt{got}(\texttt{john},\texttt{apple}),3).$$

$$\texttt{happensAt}(\texttt{go\_to}(\texttt{john},\texttt{park}),4).$$

$$\texttt{happensAt}(\texttt{got}(\texttt{john},\texttt{baseball}),5).$$

$$\texttt{happensAt}(\texttt{drop}(\texttt{john},\texttt{football}),6).$$

The next step is to provide a *definition* for the fluent `carry/2`, that is "the meaning" of carrying an object, what determines that John is carrying something with itself and

https://doi.org/10.1017/S1471068425100343 Published online by Cambridge University Press

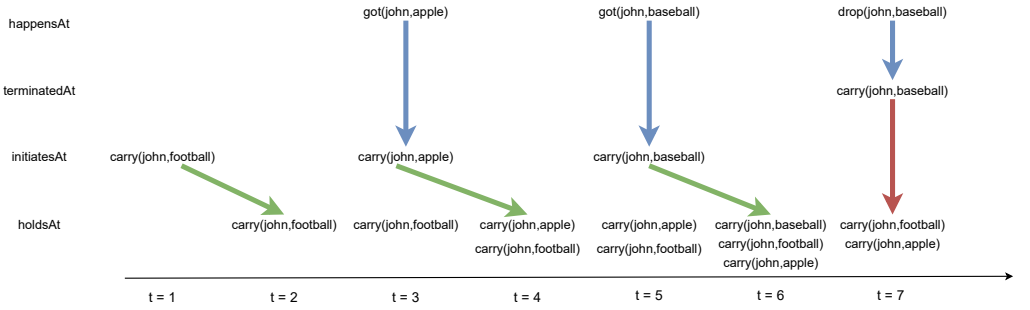

Fig 2. Fluents `carry/2` evolving over time, according to SDEC axioms. Narrative's observations – in terms of `got/2`, `drop/2` fluents – *trigger* the `carry/2` start/stop (blue arrows), which triggers `carry/2` definitions (green arrows), that dictate truth value over time due to inertia law ("something is true once it initiates and up to the point it terminates"). We can see that John carries with himself the football up to $t = 6$ when he drops it; the fluent `drop(john, football)` disables the (default) inertia rule.

when he stops doing so. Indeed, John starts carrying something with itself once he picks it up, and stops carrying something once he drops it:

$$\texttt{initiatedAt(carry(P,O),T)} : -\texttt{happensAt(got(P,O),T)}.$$

$$\texttt{terminatedAt(carry(P,O),T)} : -\texttt{happensAt(drop(P,O),T)}.$$

In this case, the commonsense knowledge that *if someone carries an item he keeps it with itself unless he drops it* is implicit in the inertial law of the second SDEC axiom. Let Π be a logic program that contains the Figure 1 rules, fluents' definitions and the narrative reified onto a set of facts as shown above. Answer sets of Π can be partitioned by the second term of each atom (which models time), and we can interpret this model as a *sequence of fluents*, as depicted in Figure 2. In this case, a single answer set is obtained. However, more complex scenarios (e.g., involving nondeterministic outcomes for actions) can be modeled by means of choice rules and constraints involving the truth value of the fluents, which might yield more than one answer set or no answer sets for the SDEC formalization, which has to be interpreted as *multiple feasible course of actions matching the narrative* or *infeasibility of the narrative (according to SDEC axioms and provided definitions)*. Consequently, ASP reasoners can be used to reason about narratives in a more complex way: checking if a given fluent is true at a given point in time, or if a desired fluent is true in all answer sets. These reasoning tasks on narratives would roughly correspond to brave reasoning and cautious reasoning in ASP.

3.3 Learning from answer sets

Inductive logic programming (ILP) (Cropper and Dumancic 2022), which aims at learning logic programs called *hypotheses* that together with an existing background knowledge explain a set of observations, has been extended to learning ASP programs (Law 2018). Learning ASP programs allows us to learn a variety of declarative non-monotonic,

https://doi.org/10.1017/S1471068425100343 Published online by Cambridge University Press

commonsense theories, including for instance the EC (Kowalski and Sergot 1986) and domain-dependent theories (Katzouris *et al*. 2015). In this paper, we use the LAS framework and its state-of-the-art system ILASP (Law 2018) for learning ASP programs. The LAS framework solves *learning tasks* which consist of a *background knowledge*, the *mode bias* and a set of *examples*. The background knowledge, denoted as B, is an ASP program which describes a set of concepts that are known before learning.

Formally, the hypothesis space H is defined as a set of (possibly non-ground) rules, and an hypothesis h is a logic program composed of rules in H, that is $h \subseteq H$. However, in ILP systems, it is not so common to explicitly provide the hypothesis space, but rather to rely on declarative means to describe it. One possible way to do so in the ILASP system is to provide the hypothesis space by means of *mode biases*.

The mode bias, denoted as M and often called *language bias*, is used to express the ASP programs that can be learned. A *mode bias* is defined as a pair of sets of mode declarations $M = \langle M_h, M_b \rangle$, where M_h (resp. M_b) are called the *head* (resp. *body*) *mode declarations*. Each mode declaration is a literal whose abstracted arguments are either `var(t)` or `const(t)`, for some constant `t` (called a *type*). For each type, a set of constants is provided along with the maximum number of variables (*maxv*) that a rule can take, thus constraining the search space induced by M. In other words, mode biases describe *what* atoms can appear in rules that will describe the hypothesis space; *maxv* acts as a filter to prune rules that contain more than a given number of variables. Informally, a literal is *compatible* with a mode declaration m if it can be constructed by replacing every instance of `var(t)` in m with a variable of type `t`, and every `const(t)` with a constant of type `t`.

The set of constants of each type is assumed to be given with a task, together with the maximum number of variables in a rule, giving a set of variables $\mathtt{V}_1, \ldots, \mathtt{V}_{\mathtt{max}}$ that can occur in a hypothesis. Whenever a variable `V` of type `t` occurs in a rule, the atom `t(V)` is added to the body of the rule to enforce the type. This guarantees the learning of safe rules.

Definition 1.
Given a mode bias $M = \langle M_h, M_b \rangle$, a normal rule R is in the hypothesis space S_M if and only if (i) the head of R is compatible with a mode declaration in M_h; (ii) each body literal of R is compatible with a mode declaration in M_b; and (iii) no variable occurs with two different types.

Example 2
(ILASP Mode Biases (Normal Rules)). In the input language of the ILASP system (Law *et al.* 2020), mode biases (for normal rules) are provided by means of the `#modeh` and `#modeb` directives. Other directives are available to express choice rules or disjunctive rules. As an example, the mode bias:

```
#modeh(a). #modeh(b).
#modeb(a). #modeb(b).
```

https://doi.org/10.1017/S1471068425100343 Published online by Cambridge University Press

states that the ground atoms `a` and `b` can belong to the head or to the body of a rule. Thus, this can be understood as a compact, declarative specifications for the set of rules[2]:

```
1 ~  :- a.
1 ~  :- b.
1 ~ b.
1 ~ a.
1 ~  :- not b.
1 ~  :- not a.
2 ~  :- a; b.
2 ~ b :- a.
2 ~ a :- b.
2 ~  :- a; not b.
2 ~ a :- not b.
2 ~  :- b; not a.
2 ~ b :- not a.
2 ~  :- not a; not b.
```

where the integer left of the tilde corresponds to the cost of the rule, that is the number of literals it contains. Thus, the provided mode biases implicitly define as hypothesis space the set of programs that is obtained by combining the above rules.

The set of examples, denoted as E, describes a set of semantic properties that the learned ASP program should satisfy. They are defined in terms of *partial interpretations*. A partial interpretation is a pair of sets of ground atoms $\langle e^{inc}, e^{exc}\rangle$, called respectively inclusion and exclusion sets. An interpretation I *extends* e iff $e^{inc} \subseteq I$ and $e^{exc} \cap I = \emptyset$. A ILASP example $ex \in E$ is a *context dependent partial interpretation* (CDPI). This is a tuple $ex = \langle ex_{id}, ex_{pi}, ex_{ctx}\rangle$, where ex_{id} is an identifier for ex, ex_{pi} is a partial interpretation and ex_{ctx} is an ASP program called a *context*. A CDPI ex is *accepted* by a program P if and only if there is an answer set of $P \cup ex_{ctx}$ that extends ex_{pi}. The idea of a context-dependent example is that each context only applies to a particular example. This is suitable for our question-answering tasks where the answer to a question is normally contextualized with respect to the story or text provided to the learner. Formally, an ILASP *context-dependent learning task* is defined as follows.

Definition 2.
A ontext-dependent Learning task (c$ILP^{context}_{LAS}$) is a tuple $T = \langle B, S_M, E\rangle$ where B is an ASP program, called the background knowledge, S_M is the set of rules allowed in the hypotheses (the hypothesis space), and E is a set of CDPIs. A hypothesis H is an inductive solution of T (written $H \in ILP^{context}_{LAS}(T)$) if and only if:

1. $H \subseteq S_M$;
2. $\forall \langle ex_{id}, ex_{pi}, ex_{ctx}\rangle \in E$, $\exists A \in AS(B \cup ex_{ctx} \cup H)$ such that A extends ex_{pi}.

[2] The output can be obtained by running the command `ILASP -s bias.lp`, where `bias.lp` is a file containing the above-specified directives.

https://doi.org/10.1017/S1471068425100343 Published online by Cambridge University Press

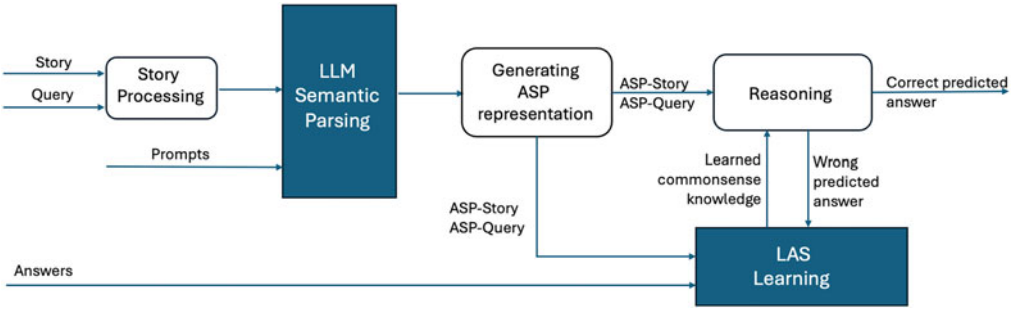

Fig 3. Architecture of LLM2LAS.

A learning task may have multiple inductive solutions. These are scored in terms of their length (i.e., number of literals they include), $score(H, T) = |H|$. An inductive solution $H \in ILP^{context}_{LAS}(T)$ is *optimal* if there is no other inductive solution $H' \in ILP^{context}_{LAS}(T)$ such that $score(H', T) < score(H, T)$.

3.4 Large language models and POS tagging

The introduction of LLM models, such as GPT and BERT, has revolutionized natural language processing (NLP) by enabling machines to process and generate human language with unprecedented accuracy (Vaswani *et al*. 2017). These deep neural network models owe their effectiveness to the transformer-based architectures (Vaswani *et al.* 2017), which utilize self-attention mechanisms to process and contextualize vast amounts of text. Most currently available LLMs have billions of parameters and are trained in a self-supervised way to predict missing tokens or the next token in a given sequence. LLMs are usually instructed through text prompts to solve a specific task, such as translating or answering questions. They have also been used successfully for semantic parsing, that is, converting text into a structured format for analysis (Nye *et al.* 2021; Drozdov *et al*. 2023; Yang *et al*. 2023).

Part-of-speech (POS) tagging involves assigning labels to tokens within a text based on their grammatical function, that is whether the token is a noun, verb, adjective, adverb, or other (Jurafsky and Martin 2009). Given a sequence $x_1, x_2, \ldots, x_n$ of words (tokens) and a set of tags, the task is to generate a sequence $y_1, y_2, \ldots, y_n$ of tags, where y_i represents the assigned tag for the input x_i. POS tagging presents challenges due to word ambiguities because a word can have multiple meanings and functions depending on the context in which it is used. In our approach, we employ the spaCy library (https://spacy.io/).

4 Methodology

In this section, we present our neuro-symbolic system LLM2LAS, which combines LLMs with LAS to learn commonsense knowledge for story-based Q&A expressed in natural language. As illustrated in Figure 3, the system consists of several modules. The *Story Processing* module normalizes the story statements and enriches them with POS tagging data, while the *LLM Semantic Parsing* generates relevant fluent and mode bias

https://doi.org/10.1017/S1471068425100343 Published online by Cambridge University Press

representations from the given story. These are then used to generate an ASP representation of the narrative described in the story. The reasoner module attempts to answer the question using the extracted narrative and the domain-independent rules given in Figure 1. If the answer is incorrect, the learner module is invoked to learn relevant commonsense knowledge from the given narrative, question, and ground-truth answer. In the following, we detail each of these steps.

Story Processing.

The module receives as input a story and a question from which the system is supposed to learn some knowledge. A story consists of an ordered set of statements describing a narrative or a scenario, while the question is designed to be answered by exploiting the information in the story. Each question is associated with the correct and incorrect answers. All sentences are *normalized* by identifying basic and compound *coreferences* – that is, whether two different expressions refer to the same entity – in the text and replacing them with their corresponding referents. Coreference resolution is automatically performed using the spaCy (Honnibal *et al.* 2020).

For example, in basic coreference, the sentence *"Mary went to the store, and she bought food."* involves replacing the word *"she"* with *"Mary."* Moreover, sentences containing negations are identified and flagged to support the following phases.

LLM Semantic Parsing.

LLMs have proven to work well in many NLP tasks, including semantic parsing (Nye *et al*. 2021; Drozdov *et al*. 2023; Yang *et al.* 2023). We exploit this strength and leverage an LLM to parse narratives and questions into fluent-like representation as in Example 1. The fluent representations support the creation of the EC representations and the mode bias declarations in the next stage.

Most available LLMs are trained on extensive public data, allowing them to achieve reasonable zero-shot generalization on diverse tasks. However, these models are not expected to perform as well in domain-specific semantic parsing tasks, where the inductive bias from pretraining is less favorable. To address this limitation, we used the few-shot prompting technique, which involves giving the model a few task-specific examples within the prompt to help guide its responses (Drozdov *et al*. 2023). Listing 1 shows an example of the prompt we have designed to ask the LLM to parse the bAbI dataset statements.

For example, if we ask an LLM model to parse the sentence *"Sam moved to the bathroom."*, using the previous prompt, the result would be: `go_to`(`sam`, `bathroom`). Our system parses each statement separately, using the same prompt multiple times to give a precise semantic representation in fluent terms. Table 2 (second column) shows a few examples of fluent representations.

Mode Bias Generation.

Mode bias fluents consist of atoms from the sentence's fluent representation where all arguments have been replaced by their types wrapped in either *"var"* or *"const."* The argument types are determined using the POS tagging data and the WH-determiners of the questions. If the sentence fluent contains an argument that is a variable (i.e., the sentence is a WH-question), then the variable is given a type in the following way: if the sentence is a *"what,"* *"when,"* or *"where"* question, then the variable's type is *"nn"*;

https://doi.org/10.1017/S1471068425100343 Published online by Cambridge University Press

Table 2. *Examples of statements with fluent and EC representations (Rep.)*

Statement	Fluent rep.	Event Calculus rep.
Mary went to the garden	`go_to(mary, garden)`	`happensAt(go_to(mary, garden), T)`
John and Helen went to the store	`go_to(john, store)`, `go_to(helen, store)`	`happensAt(go_to(john, store), T)`, `happensAt(go_to(helen, store), T)`
John is in the park or garden	{`be_in(john, park)`, `be_in(john, garden)`}	{`initiatedAt(be_in(john, park), T)`, `initiatedAt(be_in(john, garden), T)`}
Yesterday Ana went to the park	`go_to(ana, park, yesterday)`	`happensAt(go_to(ana, park, yesterday), T)`

```
Please parse the sentence provided below into a first-order logic predicate form.
The available predicates names are: go_to, be_in.
Sentence: Mary moved to the bathroom.
Semantic parse: go_to(mary,bathroom)
Sentence: John went to the hallway.
Semantic parse: go_to(john,hallway)
...

Sentence: Where is Daniel?
Semantic parse: be_in(daniel,V1)

Please, provide just the parsing data using the examples format.
The sentence to parse is:
Sentence: {{sentence}}
Semantic parse:
```

Listing. 1. Prompt for Fact Extraction.

if the sentence is a *"who"* question, then the variable's type is *"nnp"*; if the sentence is a *"why"* question, then the variable's type is *"jj"* (which stands for *adjective*), and if the sentence is a *"how many"* question, then the variable's type is *"number."* In all other cases, the argument's type is given by its associated POS tag. The types for all arguments that have a temporal aspect and the types of variables in *"why"* questions have *"const"* wrappings. The types of all other arguments are given *"var"* wrappings. The mode bias fluents aid the learner in automatically generating mode bias declarations for both formal representations. Table 3 provides an example for two narratives.

To handle this task, we introduced an LLM-based semantic parser to generate the mode bias fluents given a sentence and its fluent representation. In particular, we designed a prompt that captures the mode bias generation methodology discussed earlier and used it to request the parsing from the LLM, specifically Llama-3.3 70B. The prompts are available in the following Github repository: https://github.com/IrfanKareem/llm2las/tree/journal. This choice allowed us to overcome one of the main limitations of the previous version of the approach (Kareem *et al.* 2024), that relied on spaCy and was not general in generating the mode bias for several tasks.

https://doi.org/10.1017/S1471068425100343 Published online by Cambridge University Press

Table 3. *Sentences, fluent representations, and mode bias fluents for a short story*

Sentence	Fluent representation	Mode bias fluents
Mary is a mouse.	`be(mary, mouse)`	`be(var(nnp), var(nn))`
Mice are afraid of wolves.	`be_afraid_of(mouse, wolf)`	`be_afraid_of(var(nn), var(nn))`
What is Mary afraid of?	`be_afraid_of(mary, V1)`	`be_afraid_of(var(nnp), var(nn))`
Jason is thirsty.	`be(jason, thirsty)`	`be(var(nnp), const(jj))`
Jason went to the kitchen.	`go_to(jason, kitchen)`	`go_to(var(nnp), var(nn))`
Why did Jason go to the kitchen?	`go_to(jason, kitchen, V1)`	`go_to(var(nnp), var(nn)), const(jj))`

Generating ASP Representation.

Once statements of the story are parsed into their corresponding fluent representation, the next step is to create the EC representations (if needed). The EC representation depicts the actions and their effects in the story and comprises the four predicates introduced in Section 3. The construction of the EC representation from the fluent representation involves choosing an EC predicate and a time point. Given a sentence and its fluent representation, we select the predicate according to the following schema: (i) if the sentence is a question, then the `holdsAt/2` predicate is used; (ii) if the base of the literal's predicate is "be" and the statement is negated, then the `terminatedAt/2` predicate is used; (iii) if the base of the literal's predicate is "be" and the statement is not negated, then the `initiatedAt/2` predicate is used; (iv) otherwise, the `happensAt/2` predicate is used. The time point for the EC predicate is determined by the sentence's placement within the story. The first sentence is given time point 1, and every subsequent sentence has a time point determined by the previous one plus 1. Questions are given a time point according to when they are asked. Table 2 shows some examples of statements and their representations.

Reasoning.

The reasoning module attempts to answer a question using the information extracted from the story and the learned hypothesis. It involves automatically generating and solving an ASP program that combines the ASP representations and learned hypothesis. Ideally, the correct solution to this program (i.e., the answer sets) will contain the correct answer to the question. To extract the answer from the reasoning output, we divide the questions into two types: *"yes/no/maybe"* and others. In the case of the former, we use a *representation search* that checks the question's representation against the answer sets based on the following criteria: (i) if there is at least one answer set, and the representation is in all answer sets, then return *"yes"*; (ii) if there is at least one answer set, and the representation is in some, but not all answer sets, then return *"maybe"*; (iii) otherwise the answer is *"no."* For all other questions, we extract the answer using a *unification search*, that is by finding all ground atoms in the set of answer sets that unify with the question's formal representation. To identify these unifications, a regular expression is constructed from the question's formal representation, replacing variables with the wildcard expression *".*"*. Once unifications are detected, the ground terms corresponding

https://doi.org/10.1017/S1471068425100343 Published online by Cambridge University Press

to the *".∗"* sections of the regular expressions are added to the answer list. For example, the question in Table 3 generates *"be_afraid_of(mary, .∗)."* In case of a wrong predicted answer, the learner is invoked with the question and correct answer to learn from.

LAS Learning.

Learning commonsense knowledge from story-based Q&A is initiated through the LAS Learning module. It takes as input the EC representations of the story and the question – generated by the ASP representation module – and the correct and incorrect answers for the question. It creates the context dependent learning task for ILASP by automatically generating the mode bias declarations, using the mode bias fluent representations, and the set E of CDPI examples. To create the mode bias declarations, the system checks whether the sentence is a question, or whether the base of its fluent predicate is "be." This two-check scheme suffices to generate the language bias, given our basic sentences and limited bAbI dataset vocabulary. For questions, the system aims to learn the concept introduced in it. So, its mode bias fluent representation becomes the argument of a mode head declaration, denoted in ILASP as *"modeh."* If the sentence is not a question and the base of its fluent representation is "be," then ILASP *"modeb"* declarations are generated with the sentence's fluent as argument of mode body declaration. For the story presented in Table 3 the mode bias are:

```
#modeb(be(var(nnp),var(nn))).
#modeb(be_afraid_of(var(nn),var(nn))).
#modeh(be_afraid_of(var(nnp),var(nn))).
```

When LLM2LAS detects that the task requires reasoning about events, the language bias, referred to as EC mode bias, includes dedicated predicates. To learn the concept introduced by the question, LLM2LAS learns if it is *initiated* or *terminated*, considering also the initiation or termination of other fluents in the story. For the question's fluent, two mode head predicates are generated: *"initiatedAt"* and *"terminatedAt"* and declared as arguments of ILASP *"modeh."* A body predicate is created by enclosing the question's fluent in a *"holdsAt"* predicate, which is then wrapped in *"modeb."* For sentences that are not questions, the system detects if they describe an initiated state or a state that holds. In the first case a mode body predicate is created by enclosing the sentence's fluent into a *"initiatedAt"* predicate, in the second case the sentence's fluent is enclosed into a *"holdsAt"* predicate. Both become arguments of ILASP *"modeb"* declarations. The EC mode bias declarations for the example in Table 3 are as follows:

```
#modeb(initiatedAt(be(var(nnp),var(nn)),var(time))).
#modeb(initiatedAt(be_afraid_of(var(nn),var(nn)),var(time))).
#modeb(holdsAt(be(var(nnp),var(nn)),var(time))).
#modeb(holdsAt(be_afraid_of(var(nn),var(nn)),var(time))).
#modeh(initiatedAt(be_afraid_of(var(nnp),var(nn)),var(time))).
#modeh(terminatedAt(be_afraid_of(var(nnp),var(nn)),var(time))).
#modeb(holdsAt(be_afraid_of(var(nnp),var(nn)),var(time))).
```

https://doi.org/10.1017/S1471068425100343 Published online by Cambridge University Press

The formal representation of non-question sentences is used to prove or disprove other facts using the domain-independent EC rules in Figure 1 and the learned hypothesis. Sentences where "be"-based verbs do not appear denote actions at a specific time point in the story. Thus, their mode bias fluent representation becomes an argument of the *"happensAt"* predicate. Consider the sentence *"Mary goes to the store,"* whose mode bias fluent is `go_to(var(nnp), var(nn))`. The system generates the mode body declaration:

```
#modeb(happensAt(go_to(var(nnp),var(nn)),var(time))).
```

Another key step is the automatic generation of CDPI examples. Examples are created from questions, stories, and their correct and incorrect answers. Each example corresponds to an incorrectly answered question by the reasoning module, as this would trigger the learning module. Intuitively, the representations of all sentences prior to the question would form the example's context, the formal representations of the correct answer would be part the example's inclusion set and the formal representations of some of the incorrect answers would be part of the example's exclusion set.

We distinguish two cases, based on whether the formal presentation of the story uses choice rules (that might lead to multiple answer sets).

No Choice Rules.

In this case, only positive examples are created. The question's answer defines the example's inclusion and exclusion set: if the question answer is "yes," the question representation composes the inclusion set, otherwise it composes exclusion set. In all other cases (i.e.,, questions that are not yes/no) the inclusion set is composed with the question's correct answer, and the exclusion set is populated with a wrong answer's fluent representation.

Choice Rules.

If choice rules are present in the formal representation of a story, the example generation has to take into account brave and cautious entailment. For a yes/no/maybe question, if the answer is "maybe," then the example will include the question representation in its inclusion set to guarantee that the question's concept occurs in at least one answer set. If the correct answer is a "yes," then a negative example is created where the inclusion set is empty and the exclusion set includes the question's correct answers. This is to guarantee that the question's formal representation is true in all answer sets. In all other cases, a negative example is created with an empty exclusion set and inclusion set given by the representation of the question's answers. This is to guarantee the wrong answer will be false in all answer sets.

To illustrate some of the cases explained above, consider the following story: *Daniel went to the kitchen. Daniel went to the bedroom.* The question: *Where is Daniel?* The correct answer is the *bedroom* and incorrect answer is *kitchen*. If the incorrect answer is predicted, then the following example is created:

https://doi.org/10.1017/S1471068425100343 Published online by Cambridge University Press

```
% Background Knowledge
holdsAt(F,T+1) :- initiatedAt(F,T),time(T).
holdsAt(F,T+1) :- holdsAt(F,T), not terminatedAt(F,T),time(T).

% Mode bias
#modeb(happensAt(go_to(var(nnp),var(nn)),var(time))).
#modeb(holdsAt(be(var(nnp),var(nn)),var(time))).
#modeh(initiatedAt(be(var(nnp),var(nn)),var(time))).
#modeh(terminatedAt(be(var(nnp),var(nn)),var(time))).

% Positive and Negative Examples
#pos({holdsAt(be(daniel,bedroom),3)},{holdsAt(be(daniel,kitchen)
,3)},{time(1..3). happensAt(go_to(daniel,kitchen),1).
happensAt(go_to(daniel,bedroom),2).}).
```

The system requires minimal background knowledge. If the EC is required, the background knowledge will consist of the rules in Figure 1, that encode the notion of *inertia* (e.g., if a has initiated previously before time t, and has not been terminated, it continues to hold at time $t+1$). When the ILASP system is run to solve the generated learning task, if the task is satisfiable, the reasoner is updated with the learned hypothesis.

5 Empirical evaluation

In this section, we report on our evaluation of the proposed approach on a well-known Q&A dataset. To this end, we first describe the bAbI dataset from Facebook Research (Weston *et al*. 2016), and then provide a description of the hardware and software configurations we have employed and of our baselines. Finally, we comment on the results that confirm the efficacy of our system.

5.1 Experiment setup

Dataset.

The bAbI dataset is composed of 20 non trivial tasks of text understanding and reasoning that was proposed by Facebook Research. The bAbI dataset was conceived as a benchmark for assessing a range of natural language reasoning abilities, including deduction, path finding, spatial reasoning, and counting (Weston *et al*. 2016). Each task of the dataset is constructed by simulating words that represent entities and actions. An entity, denoted as a noun, can be a location, an object, or a person, and possesses internal attributes such as size, color, or relative position to cardinal directions. Within this simulation, each entity can perform ten fundamental actions, with each action associated with a collection of replacement synonyms, pronouns, and temporal adverbs, ensuring lexical diversity within the tasks. More in detail, the dataset comprises 4 actors, 6 locations, and 3 objects per task, it features stories ranging from 3 to 229 sentences and 1 to 12 questions. Each sentence within a story is uniquely identified and accompanied by its answer for each task. The dataset includes both training and test data for each task, with a strong focus on learning from a few examples. The stories are also available in human-readable formats in various languages. In the experiment, we place our focus on

https://doi.org/10.1017/S1471068425100343 Published online by Cambridge University Press

Table 4. *Tasks of the bAbI dataset. "Solved w/t" stands for "solved with impr."*

Task name	Status	Requires EC?	Notes
(1) Single Supporting Fact	Solved	✓	
(2) Two Supporting Facts	Unsolved		Additional mode bias declarations required and large hypothesis space
(3) Three Supporting Facts	Unsolved		Additional mode bias declarations required and large hypothesis space
(4) Two Argument Relations	Solved		
(5) Three Argument Relations	Unsolved		ILASP hypothesis space too large
(6) Yes/No Questions	Solved	✓	
(7) Counting	Solved w/t	✓	Counting background knowledge added
(8) Lists/Sets	Solved	✓	
(9) Simple Negation	Solved	✓	
(10) Indefinite Knowledge	Solved	✓	
(11) Basic Coreference	Solved	✓	
(12) Conjunction	Solved	✓	
(13) Compound Coreference	Solved	✓	
(14) Time Reasoning	Solved		
(15) Basic Deduction	Solved	✓	
(16) Basic Induction	Solved		
(17) Positional Reasoning	Solved w/t		Recursion removed from learned program
(18) Size Reasoning	Solved		
(19) Path Finding	Solved		
(20) Agents Motivations	Solved		

the natural language, examining the tasks for which our implementation can be applied. The considered tasks are detailed in Table 4.

Hardware and Software Setup.

All experiments are conducted on a computer equipped with a AMD EPYC 7313 16-Core processor, 2 TB of RAM, and GPU AMD Instinct MI210 with 64 GB of memory. The experiment pipelines are implemented in the Python programming language version 3.9. Our architecture has been implemented using the open-source LLM LLama-3.3 70b, Clingo 5.6.2 for reasoning on ASP programs, and ILASP 4.4.1 for LAS. In particular, we use the `2i` version of the ILASP system, which has proved to be the most suitable for our purposes due to the incremental processing of examples. Concerning the learning parameters, the maximum penalty for the size of the hypothesis was set to 50, and the maximum number of variables was set to 3 for the tasks solved with fluent representation, and to 4 for the tasks solved with EC representation. Each task was evaluated on 1000 training examples.[3] For each considered task, we measure the accuracy (e.g., ratio over correct answers) of compared methods.

[3] The values for these parameters has been determined experimentally. Tasks that use the EC require one extra variable due to the *time* variable that appears in the background knowledge.

https://doi.org/10.1017/S1471068425100343 Published online by Cambridge University Press

Our implementation has been also compared against two baselines from the literature, also based on logic programming: the ILP-based system in Mitra *et al.* (Mitra and Baral 2016), and the approach proposed by Yang *et al.* (2023).

All the material needed to reproduce our experiments can be downloaded from: https://github.com/IrfanKareem/llm2las/tree/journal.

5.2 Results and discussion

We discuss our results in three separate paragraphs, considering three different settings. The first paragraph assesses the system applying – without any expert knowledge intervention – the workflow in Figure 3; the second describes some techniques we applied to the basic workflow to improve its performance; the third paragraph focuses on identifying the cases where our system had some difficulty; finally, we compare it with alternative solutions. The section concludes with a general discussion summarizing our findings.

Table 4 summarizes the results obtained while approaching the various tasks in the dataset. Each row in the table reports the task number and name, the status of the task, a flag indicating whether the EC background knowledge is needed, and a short note about improvements made to the basic pipeline (if any).

Tasks Solved within the Framework.

First of all, we report that LLM2LAS could be applied to all bAbI tasks, correctly generating ASP representation for stories and mode biases. Then, the system was able to learn in a reasonable time (within 24 hours) the ASP specification for 15 tasks out of 20. In particular, the solved tasks required on average 40 s, for a cumulative learning time of 9 minutes. The bAbI tasks that were fully solved are: 1, 4, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 18, 19, and 20. Of these 4, 14, 16, 18, 19, and 20 required no background knowledge, and the remaining, namely 1, 6, 8, 9, 10, 11, 12, 13, and 15, required EC background knowledge. LLM2LAS achieved perfect accuracy (100% accuracy), that is, it was able to learn an ASP program solving task without any human intervention. The average runtime for learning is a matter of a few seconds, once ILASP generates the hypothesis space for the first time. Indeed, the hypothesis space is cached and reused across multiple examples in an incremental learning setup, significantly improving efficiency.

Mitigating Hypothesis Space Size.

LLM2LAS struggled in the remaining 5 tasks because the ILP task was too expensive, often because of the size of the hypothesis space. Thus, upon inspecting its cause, we applied some ad hoc improvements on a task-by-task basis to prune the size of the hypothesis, such as adding background knowledge involving aggregates for arithmetic-related reasoning, learning non-recursive programs and marking some predicates as being symmetric or anti-symmetric. In this way we solved bAbI Tasks 7 and 17 with an intervention on the learning tasks.

Task 7 involves basic arithmetic reasoning, specifically the ability to track the number of items an individual is carrying based on a sequence of actions described in the narrative. Here, it is required to learn how to count items, but ILASP cannot learn effectively

https://doi.org/10.1017/S1471068425100343 Published online by Cambridge University Press

programs that use aggregates. Nonetheless, counting can be considered basic knowledge, so we added the following rule to the background knowledge:

$$\texttt{carriedItems(X,N,T):-holdsAt(carry(X,\_),T),N} = \texttt{\#count\{Z : holdsAt(carry(X,Z),T)\}}.$$

that is, we provide an explicit definition for the "number of items carried by a person at a given point in time." Including this rule makes the task solvable by our system, which is able to learn the initiating and terminating conditions for the `carry/2` fluent. Thus, we obtained 100% accuracy for this task, with a learning time of around 46 s.

On the other hand, Task 17 deals with positional reasoning and, in particular, the relative positioning of objects in a scene, for example learning the definition of "being left of something," and "right of something," modeled by means of `be_right_of/2`, `be_left_of/2`, `be_above_of/2`, `be_below_of/2` atoms. Informally, the learning task involves acquiring both the knowledge that spatial relations such as left–right and above-below are opposites and "symmetric," as well as learning rule pairs that define the transitive closure of these spatial predicates. As an example, the for `be_above_of/2` predicate we should have:

```
% above, below are antisymmetric
be_above_of(X,Y) :- be_below_of(Y,X).
% transitive closure of the be_above_of/2 predicate
be_above_of(X,Y) :- be_above_of(X,Z), be_above_of(Z,Y).
```

We observed that the learning task can be made less heavy by reformulating it so that a non-recursive solution is admitted. This can be obtained by introducing auxiliary predicates (limited to the head of rules) of the form `be_*/2` for each `be_*_of/2` predicate.

Although with this improvement we managed to learn an ASP specification that covers most of the examples, thus circumventing the learning bottleneck, this solution does not generalize as the intended (recursive) solution. In particular, the system obtained an accuracy of 97.8%, with a learning time of 16 minutes on average.

Challenges and Open Tasks.

For the remaining tasks, namely 2, 3, and 5, there was no space for reducing the impact of the learning phase.

Task 5 consists of learning narratives that involve tracking movement, location and possession of objects over time. It is characterized both by the requirement of EC background knowledge, and the need for ternary fluents ($\texttt{give\_to}(\texttt{P}_1,\texttt{P}_2,\texttt{O})$–$P_1$ has given object O to P_2, $\texttt{receive}(\texttt{P}_1,\texttt{P}_2,\texttt{O})$–$P_1$ has received object O from P_2). LLM2LAS correctly generates the learning task, but the joint presence of ternary fluents and EC yields a too large hypothesis space, that the ILASP system is unable to ground (i.e., the ILASP system could not build the set of rules that can appear in an hypothesis) in a reasonable time.

Task 2 consists of narratives that involve tracking the position of objects over time; and Task 3 is a more complex version of Task 2. In these cases, the primary challenge arises from the need to learn multiple commonsense notions, such as understanding that

https://doi.org/10.1017/S1471068425100343 Published online by Cambridge University Press

an agent is "in" a location after moving, or that receiving a gift implies "possession" or "ownership," that are not explicitly stated within the narrative. Being able to process this learning task without substantial additions to the background knowledge remains an open problem.

Comparison with Other Systems.

We now compare LLM2LAS with existing approaches in the literature both in terms of accuracy and in terms of the need for human intervention on the 17 successfully-solved tasks. For the systems by Mitra and Baral (2016),Yang *et al.* (2023) we use the accuracy reported in the respective publications. The approach of Yang *et al.*, which relies on humanly-devised ASP programs, reaches 100% accuracy in all 17 tasks. On the other hand, the approach by Mitra and Baral can obtain 100% accuracy in all tasks but task 16 (where accuracy is of 93.6%). Finally, LLM2LAS achieves 100% accuracy in all tasks but task 17 where it achieves 97.8% of accuracy.

Although the performance of the compared methods are essentially aligned in terms of accuracy, there is a major difference that has to be outlined: our approach does not require writing ASP code, which is learned automatically from the examples in the training sets of the bAbI dataset.

6 Discussion

The results reported above show that LLM2LAS successfully learns, reasons and provides answers over 17 commonsense-driven tasks in the bAbI dataset, matching the performance of most of the human-expert manually engineered ASP programs. Although this is a promising result in combining LLMs with logic reasoners, the experiment also helped to identify two open problems: (*i*) there are tasks we can in principle solve, but the ILP task is out of reach for the learning system (due to size of generated hypothesis space); and (*ii*) the approach struggles to deal with notions that are not explicitly mentioned within the narrative.

Focusing on issue (*ii*), we observe that while there is a growing consensus that LLMs encode a certain degree of commonsense knowledge about the world, the framework proposed in this work does not currently leverage this capability to pre-populate the background knowledge with a set of task-relevant rules. Enabling such integration represents a promising direction for future research, aligning well with current trends in neuro-symbolic learning and the broader effort to bridge statistical and symbolic reasoning.

7 Conclusion

This work presents LLM2LAS, a novel hybrid framework that advances the integration of LLMs with symbolic reasoning, by introducing an automated pipeline for learning commonsense knowledge from examples. Building on prior research that combines LLMs with symbolic components for story-based Q&A, our approach moves a step forward existing methods by eliminating the need for manually crafted logic rules. This is obtained by leveraging LLMs for semantic parsing, ILASP for rule induction, and ASP for reasoning.

https://doi.org/10.1017/S1471068425100343 Published online by Cambridge University Press

Our results on the bAbI dataset demonstrate that it is not only feasible but also effective (in terms of accuracy) to automatically induce ASP specifications with minimal supervision, thus reducing the reliance on human modeling expertise. From an accuracy standpoint, LLM2LAS is capable of matching solutions based on manually crafted ASP encodings. At the same time, our findings highlight limitations of our implementation: current LAS systems, such as ILASP, sometimes struggle with scalability when faced with large or complex hypothesis spaces, and with notions that are not explicitly mentioned in the datasets. Another limitation lies in the support of mathematical reasoning constructs, as at the time being ILASP cannot learn programs with aggregates, whereas LLMs are able to extract this kind of information in the semantic parsing step. However, knowledge engineers can mitigate these issues by providing more background knowledge, or by resorting to hypothesis space pruning techniques.

Future works will explore methods to further address this bottleneck, potentially through hypothesis space pruning techniques, automated background knowledge extraction, support for richer mathematical reasoning constructs (e.g., aggregates) and tighter LLM-LAS integration, as well as alternative learning strategies. Overall, LLM2LAS contributes a promising step toward more autonomous, interpretable, and robust systems for reasoning in natural language tasks.

Acknowledgments

This work was partially supported by the Italian Ministries MIMIT, under project EI-TWIN n. F/310168/05/X56 CUP B29J24000680005, project ASVIN n. F/360050/01-02/X75 CUP B29J2400020000, and MUR, under projects: PNRR FAIR – Spoke 9 – WP 9.1 CUP H23C22000860006, Tech4You CUP H23C22000370006, and PRIN PINPOINT CUP H23C22000280006.

Competing interests

The author(s) declare none.

References

Al-Negheimish, H., Madhyastha, P. and Russo, A. 2021. Numerical reasoning in machine reading comprehension tasks: are we there yet?. In *EMNLP (1)*. Association for Computational Linguistics, 9643–9649.

Alviano, M. and Grillo, L. 2024. Answer set programming and large language models interaction with YAML: Preliminary report. In CILC. CEUR Workshop Proceedings, vol. 3733. CEUR-WS.org.

Alviano, M., Scudo, F. L., Grillo, L. and Reiners, L. A. R. 2024. Answer set programming and large language models interaction with YAML: Second report. In KoDis+CAKR+SYNERGY@KR. CEUR Workshop Proceedings, vol. 3876. CEUR-WS.org.

Apt, K. R. 2003. *Principles of Constraint Programming*. Cambridge University Press.

Brewka, G., Eiter, T. and Truszczynski, M. 2011. Answer set programming at a glance. *Communications of the ACM* 54, 12, 92–103.

https://doi.org/10.1017/S1471068425100343 Published online by Cambridge University Press

Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Maratea, M., Ricca, F. and Schaub, T. 2020. Asp-core-2 input language format. *Theory and Practice of Logic Programming* 20, 2, 294–309.

Cheng, F., Li, H., Liu, F., van Rooij, R., Zhang, K. and Lin, Z. 2025. Empowering llms with logical reasoning: A comprehensive survey. CoRR abs/2502.15652.

Colmerauer, A. and Roussel, P. 1993. The birth of prolog. In *HOPL Preprints*. ACM, 37–52.

Cropper, A. and Dumancic, S. 2022. Inductive logic programming at 30: A new introduction. *Journal of Artificial Intelligence Research* 74, 765–850.

Davis, E. and Marcus, G. 2015. Commonsense reasoning and commonsense knowledge in artificial intelligence. *Communications of the ACM* 58, 9, 92–103.

Drozdov, A., Schärli, N., Akyürek, E., Scales, N., Song, X., Chen, X., Bousquet, O. and Zhou, D. 2023. Compositional semantic parsing with large language models. In *ICLR*. OpenReview.net.

Gebser, M., Kaminski, R., Kaufmann, B. and Schaub, T. 2019. Multi-shot ASP solving with clingo. *Theory and Practice of Logic Programming* 19, 1, 27–82.

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In *ICLP/SLP*, vol. 88, 1070–1080.

Honnibal, M., Montani, I., Van Landeghem, S. and Boyd, A. 2020. spaCy: Industrial-strength Natural Language Processing in Python.

Ishay, A., Yang, Z. and Lee, J. 2023. Leveraging large language models to generate answer set programs. In *KR*, 374–383.

Jurafsky, D. and Martin, J. H. 2009. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 2nd ed. Prentice Hall, Pearson Education International.

Kareem, I., Gallagher, K., Borroto, M. A., Ricca, F. and Russo, A. 2024. Using learning from answer sets for robust question answering with LLM. In *LPNMR*. Lecture Notes in Computer Science, vol. 15245. Springer, 112–125.

Katzouris, N., Artikis, A. and Paliouras, G. 2015a. Incremental learning of event definitions with inductive logic programming. *Machine Learning* 100, 2–3,555–585.

Kautz, H. A. 2022. The third AI summer: AAAI robert s. engelmore memorial lecture. *AI Magazine* 43, 1, 93–104.

Kim, J., Kim, D. and Yang, Y. 2024. Learning to correct for QA reasoning with black-box llms. In *EMNLP*. Association for Computational Linguistics, 8916–8937.

Kowalski, R. and Sergot, M. 1986. A logic-based calculus of events. *New Generation Computing* 4, 67–95.

Lake, B. M. and Murphy, G. L. 2020. Word meaning in minds and machines. CoRR abs/2008.01766.

Law, M. 2018. Inductive Learning of Answer Set Programs. Ph.D. thesis, London, UK

Law, M., Russo, A. and Broda, K. 2015. Simplified reduct for choice rules in ASP. Tech. Rep. DTR2015-2, Imperial College of Science, Technology and Medicine, Department of Computing.

Law, M., Russo, A. and Broda, K. 2020. The ILASP system for inductive learning of answer set programs. CoRR abs/2005.00904.

Lifschitz, V. 2008. What is answer set programming?. In *AAAI*. AAAI Press, 1594–1597.

Luo, M., Kumbhar, S., Shen, M., Parmar, M., Varshney, N., Banerjee, P., Aditya, S. and Baral, C. 2023. Towards logiglue: A brief survey and A benchmark for analyzing logical reasoning capabilities of language models. CoRR abs/2310.00836.

https://doi.org/10.1017/S1471068425100343 Published online by Cambridge University Press

Mitra, A. and Baral, C. 2016. Addressing a question answering challenge by combining statistical methods with inductive rule learning and reasoning. In *AAAI*. AAAI Press, 2779–2785.

Nye, M. I., Tessler, M. H., Tenenbaum, J. B. and Lake, B. M. 2021. Improving coherence and consistency in neural sequence models with dual-system, neuro-symbolic reasoning. In *NeurIPS*, 25192–25204.

Ray, O. 2009. Nonmonotonic abductive inductive learning. *Journal of Applied Logic* 7, 3, 329–340.

Ruis, L., Andreas, J., Baroni, M., Bouchacourt, D. and Lake, B. M. 2020. A benchmark for systematic generalization in grounded language understanding. In *NeurIPS*.

Sap, M., Shwartz, V., Bosselut, A., Choi, Y. and Roth, D. 2020. Commonsense reasoning for natural language processing. In *ACL (tutorial)*. ACL, 27–33.

Shanahan, M. 1999. The event calculus explained. In *Artificial Intelligence Today*. Lecture Notes in Computer Science, vol. 1600. Springer, 409–430.

Sinha, K., Sodhani, S., Dong, J., Pineau, J. and Hamilton, W. L. 2019. CLUTRR: A diagnostic benchmark for inductive reasoning from text. In *EMNLP/IJCNLP (1)*. Association for Computational Linguistics, 4505–4514.

Tafjord, O., Dalvi, B. and Clark, P. 2021. Proofwriter: Generating implications, proofs, and abductive statements over natural language. In *ACL/IJCNLP (Findings). Findings of ACL, ACL/IJCNLP 2021*, 3621–3634.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. and Polosukhin, I. 2017. Attention is all you need. In *NIPS*, 5998–6008.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. H., Le, Q. V. and Zhou, D. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*.

Weston, J., Bordes, A., Chopra, S. and Mikolov, T. 2016. Towards ai-complete question answering: A set of prerequisite toy tasks. In *ICLR (Poster)*.

Wu, Y., Han, X., Song, W., Cheng, M. and Li, F. 2024. Mindmap: Constructing evidence chains for multi-step reasoning in large language models. In *AAAI*. AAAI Press, 19270–19278.

Yang, Z., Ishay, A. and Lee, J. 2023. Coupling large language models with logic programming for robust and general reasoning from text. In *ACL (Findings)*. Association for Computational Linguistics, 5186–5219.

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y. and Zhang, B., *et al.* 2023. A survey of large language models. arXiv preprint arXiv:2303.18223.

Zheng, S., Huang, J. and Chang, K. C.-C. 2023. Why does chatgpt fall short in answering questions faithfully? arXiv preprint arXiv:2304.10513.

https://doi.org/10.1017/S1471068425100343 Published online by Cambridge University Press