LAMP: ASP Configuration Model Rules Deduction and Optimization with LLMs

David Bunker

University of Potsdam

Abstract. Answer Set Programming (ASP) offers a compelling mechanism to represent product configuration. However, being able to deduce such programs that are sufficiently performant remains challenging for many users. In this area Large Language Models (LLMs) have the potential to bridge this gap. This work proposes a LLM to ASP Modeling Protocol (LAMP) system to deduce and optimize ASP rules based on provided instance facts and the expected stable models. This system also facilitates the evaluation of LLM capabilities in this regard of which gpt-oss, gpt-5-mini and qwen3-coder where selected for testing. To properly perform the evaluation a variety of ASP program complexities and potential optimizations were assessed including: varying the number of rules, number of positive and negative body literals, and number of predicates and terms. The evaluation of varying complexities also demonstrates which program features are most challenging for LLMs in ASP program generation.

1 Introduction

Product configuration problems arise across many domains, including software and hardware customization, industrial manufacturing, service composition among others. These problems are characterized by combinatorial constraints that determine which combinations of components are valid. Answer Set Programming (ASP) models such problems, offering clear semantics based on stable models with expressive capabilities for constraints, defaults, and combinatorial reasoning.

However, writing ASP programs that are both correct and performant remains challenging, particularly for new users, as modeling choices can significantly affect solver behavior. Recent progress in Large Language Models (LLMs) offers new opportunities to reduce this modeling burden. LLMs have shown strong capabilities in code generation and program synthesis, suggesting they may assist in deriving ASP encodings from high level specifications [11].

LLMs can also be evaluated for logic programming abilities using such systems. It has been shown LLMs can perform inductive logic programming, such as learning logic rules from background knowledge and positive and negative examples when combined with feedback from a formal inference engine, such as Prolog [7]. Here the authors proposed a systematic evaluation framework graded by expressivity to quantify LLM strengths and limitations in logic theory induction.

Similarly, the proposed LLM to ASP Modeling Protocol (LAMP) is designed to guide LLMs in deducing and optimizing ASP rules from given instance facts and expected stable models. LAMP provides a structured interaction protocol that enables refinement of ASP encodings, with the goal of improving both correctness and performance. Beyond proposing the system itself, LAMP is used as a framework for evaluating the capabilities of different LLMs in ASP program generation.

The relevant research questions of this work are:
RQ1. Is a structured approach for LLM assisted ASP modeling possible using LAMP system?
RQ2. Does this system allow for comparative evaluation of LLMs on ASP generation and optimization tasks?
RQ3. Which ASP language features most strongly affect LLM performance?

This proposal investigates the current capabilities of Large Language Models (LLMs) regarding Answer Set Programming (ASP) and offers directions for future research at the intersection of artificial intelligence and constraint solving. Such a system can then be contrasted against similar systems such as those based on LLMs fine-tuned to ASP [3], iterative scheduling and plan generation with LLMs and ASP [16], and LLM based systems that can learn directly from ASP answer sets [1].

2 ASP, Complexity and Datasets

This section presents a method for applying ASP to product configuration that is suited for use with LLMs. This is followed by several ways to systematically vary the complexity, enabling an analysis of the LAMP process and prompting quality, LLM performance, and the specific features that pose the greatest challenges for LLMs. Finally, the process used to generate datasets of ASP programs for these experiments is outlined.

2.1 Answer Set Programming

ASP is a declarative programming language based on logic programming and non-monotonic reasoning where previous conclusions can be invalidated by new information. Instead of individual instructions, programs are described with logical rules and constraints and an ASP solver computes the solutions as stable models, referred to as answer sets, that satisfy them.

There are many ways to represent product configuration using ASP, one such example being *OOASP* [6], offering an expressive object-oriented product configuration system. For use as a demonstration of LLM evaluation and ASP program generation and optimization a simpler approach is more suitable option. This simplification includes ASP *facts* of form `descriptor(object)` with possible *normal rules* `new_descriptor(X) :- descriptor_0(X), [not]` descriptor_1(X), ... allowing new object descriptions to be resolved from initial instance object descriptions. These ASP program constraints offer flexibility without overwhelming the LLMs with semantic possibilities.

Below is an example representing the model facts that objects "car" and "scooter" are both vehicles, represented with the predicate vehicle, and the scooter is slow moving, represented with the predicate slow_moving. It also has the rule that if an object is a vehicle and not slow moving it is highway possible, represented with the predicate highway_possible.

```
vehicle("scooter").
vehicle("car").
slow_moving("scooter").
highway_possible(X) :- vehicle(X), not slow_moving(X).
```

Running this results in the answer set { vehicle("bike"), vehicle("car"), slow_moving("scooter"), highway_possible("car") }. Given these facts and rules, highway_possible(car) can be deduced, indicating the car is highway possible and, due to default negation, the scooter is not highway possible.

2.2 Complexity

There are many ways to evaluate complexity of an ASP program. In the paper *Inductive Learning of Logical Theories with LLMs*, which investigates LLM based Prolog rule generation, this is approached as categories Chain, Rooted Directed Graph, Disjunctive Rooted Directed Graph, and Mixed [7]. The Chain category is simplest in that every rule, except the root, deduces facts based on only one other rule, Rooted Directed Graph extends this where every rule can be relevant for several others by its head appearing in their bodies. Disjunctive Rooted Directed Graph extends this further by predicates appearing as head of multiple rules, facilitating disjunction. Mixed is the most complicated as it is a combination of each in addition to recursion, where a rule's head predicate can also appear in its body.

As this work focuses specifically on correct answer set deduction and rule optimization, fact and rule features are used to represent complexity. These features include, number of possible predicates, terms, facts, rules, positive literals within each rule, and negative literals within each rule. These could be extended to include greater arity, or additional language features, such as choice rules and head disjunction for future work.

Another approach for future work would be from the specific perspective of solving complexity, focussing on solution space features such as, vector cover size, tree depth, feedback vertex set, clique width, tree width, among others [9].

2.3 Dataset Generation

To test against the complexity principles proposed, synthetic data is generated by introducing new possible predicates and terms as needed. To keep the programs sufficiently complex, but manageable by the LLMs, only one stable model is expected and at least one atom must be deduced. The vehicle example above would be represented as shown below.

```
d0(o0).
d0(o1).
d1(o1).
d2(X) :- d0(X), not d1(X).
```

This results in the answer set `{ d0(o0), d0(o1), d1(o1), d2(o0) }`, corresponding to the vehicle example answer set.

3 Methodology

The ASP rule generation procedure functions by iteratively prompting the LLM, providing the facts and expected answer set.

3.1 LLM Prompting

LLM prompting has been shown to be effective in writing ASP programs in such areas as logic puzzle solving [15] and robotic task planning [16]. LLM and ASP based hybrid systems have even been shown to be effective in improving natural language understanding [19], common sense reasoning [25] and as collaborative conversations agents [24].

The prompt provided to the LLMs provides the instance atoms facts, such as `descriptor_0(object_0)`, expected ASP output format `predicate_0(X) :- predicate_1(X), [not] predicate_2(X)...`, and the expected stable model, such as `d0(o0), d0(o1), ...`. The output is then parsed into ASP and run with the ASP solver Clingo. This can then be passed back if incorrect.

As future work the prompt may be able to be optimized by adding examples or using DSPy [14] which tests many variations of the prompt with different examples to get the best result. This was demonstrated for spatial reasoning as a chain of thought reasoning system [22]. Another option would be to fine tune the LLM specifically for ASP code generation as was done with LLASP [3].

3.2 Evaluation

The ASP programs generated are evaluated after being run via Clingo. The number of rules, complexity and how close it is to the expected stable model can then be assessed. This is similar to the process LLM-ARC employs as an actor-critic method, where the LLM Actor generates ASP, while the automated reasoning critic evaluates the code [12]. Other systems use ASP as a scaffolding for robust reasoning [13].

4 Experiments

Experiments where conducted on the synthetic data with varying complexities as specified in the section *ASP, Complexity and Datasets*. A variety of online and local open weight LLMs where selected for testing to get a broad view of capabilities.

4.1 Experimental Setup

A total of 120 synthetic ASP programs where created with varying features for testing. The LLMs chosen included *gpt-5-mini*, which was run via OpenAIs API platform, as well as open weight models *gpt-oss:20b* and *qwen3-coder:30b* which where run locally using Ollama. All ASP solving and analysis was done using Clingo.

4.2 Results and Discussion

Overall results are shown in table 1. Both *gpt-5-mini* and *gpt-oss:20b* perform very well, deducing the correct ASP rules to get the expected stable model in almost all cases. *qwen3-coder:30b* performed significantly worse, deducing the correct rules in almost all cases. Although *qwen3-coder:30b* is tuned for programming, it may not have been tuned for ASP, resulting in poorer performance. Of the cases the correct stable model was produced, most of the time the rules did not match the ones from the synthetic data.

LLM	Correct	Rules Matching
gpt-5-mini	119	11
gpt-oss:20b	116	11
qwen3-coder:30b	53	1

LLM	Incorrect
gpt-5-mini	1
gpt-oss:20b	4
qwen3-coder:30b	67

Table 1: LLM with Answer Set Correct and Incorrect Results, Rules Matching Indicates number of LLM Generated Programs that Match the Original.

Table 2 shows the number of rules of deduced by the indicated LLM subtracted from normal of rules from the original synthetically generated data. The left table is for when the stable model is correct and the left is for incorrect. The rule count matches much of the time for both `gpt-5-mini` and `gpt-oss:20b`, but often optimizes to fewer rules reducing by as many as 2 rules 24 times for both. `qwen3-coder:30b` performs worse, creating more rules than the original in 10 cases.

Figures 1, 2 and 2 compare the total number of literals in the original ASP against the total number of literals of the LLM generated rules for each LLM tested. The dot size indicates the number of examples with left for when the LLM

LLM	Rules Diff	Correct
gpt-5-mini	0	59
gpt-5-mini	1	36
gpt-5-mini	2	24
gpt-oss:20b	0	58
gpt-oss:20b	1	34
gpt-oss:20b	2	24
qwen3-coder:30b	-2	3
qwen3-coder:30b	-1	7
qwen3-coder:30b	0	17
qwen3-coder:30b	1	15
qwen3-coder:30b	2	10

LLM	Rules Diff	Incorrect
gpt-5-mini	0	1
gpt-oss:20b	0	2
gpt-oss:20b	1	2
qwen3-coder:30b	-2	9
qwen3-coder:30b	-1	11
qwen3-coder:30b	0	13
qwen3-coder:30b	1	13
qwen3-coder:30b	2	10

Table 2: Number of Correct and Incorrect Answer Set Results by LLM and Number of Rules Fewer than Original (Larger is Better).

got the correct stable model and right for incorrect. Figure 2 shows `gpt-5-mini` is generally able to greatly reduce the number of literals needed, in some case reducing number of literals by 6, but in some cases increasing them by 2. LLM `qwen3-gpt_oss_literals:20b` shown in figure 2 had similar results generally needing even fewer literals, while `qwen3_literals` shown in figure 3 had mixed results. Overall, the results corroborate with other benchmarking studies on the abilities of LLMs to generate ASP [20].

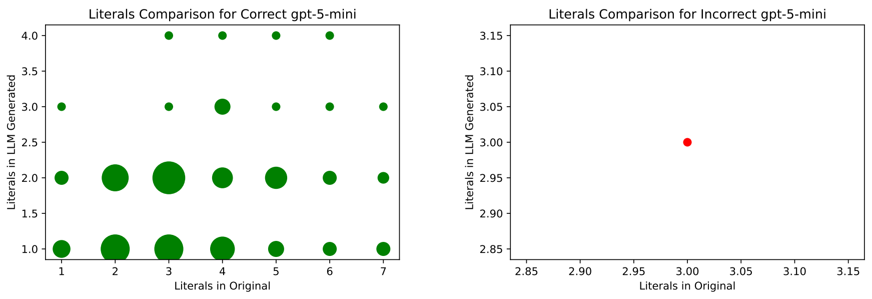

Fig. 1: Gpt-5-mini input literals to output literals for correct answer set response.

4.3 Feature Importance

Based on whether the LLM is able to get the correct stable model or not it is possible to deduced which ASP rule features are most difficult for LLMs to deduce thereby indicating higher complexity. To do this, the features of the

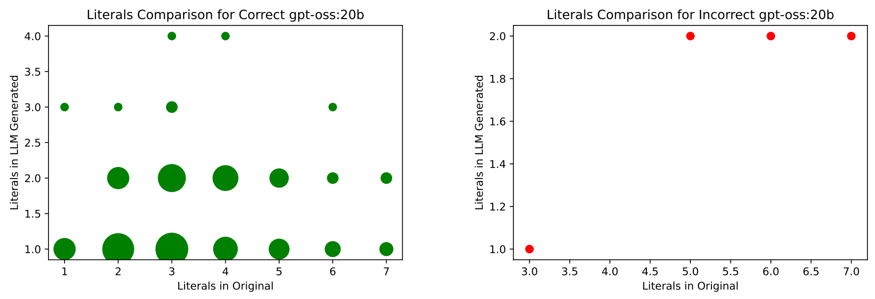

Fig. 2: Gpt-oss:20b input literals to output literals for correct answer set response.

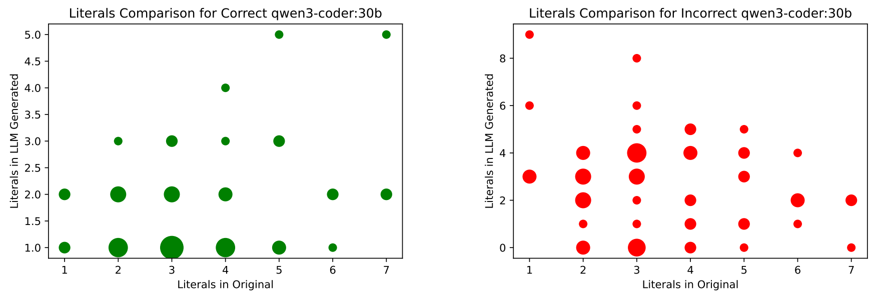

Fig. 3: Qwen3-coder input literals to output literals for correct answer set response.

original ASP program and the LLM deduced ASP program such as number of rules, number of new atoms inferred etc. are used to predict the likelihood of a getting the correct stable model using the tree based machine learning algorithm XGBoost.

From the XGBoost model the importance value of each feature can be calculated. There are multiple ways to get importance, such as gain, or average improvement in the loss function from splits using that feature or cover, the average number of samples affected by splits using that feature. For this analysis weight is used, the number of times a feature is used to split across all trees [17].

The results can be seen in figure 4. The most important feature according to this analysis is *atom_diff_llm*, or the number of new atoms infered by the LLMs program. This makes sense as the LLMs ASP not inferring atoms is a good indicator it won't arrive at the expected stable model. The next most important feature is *total_neg_literals_llm*, indicating that having a lot of negative literals adds complexity.

A similar method that is somewhat more robust involves deriving the features SHAP values. Each feature's SHAP value is the average change in prediction caused by that feature taken over all possible ways that feature could have appeared in the model [17]. SHAP values shown in figure 5 show similar results to figure 4.

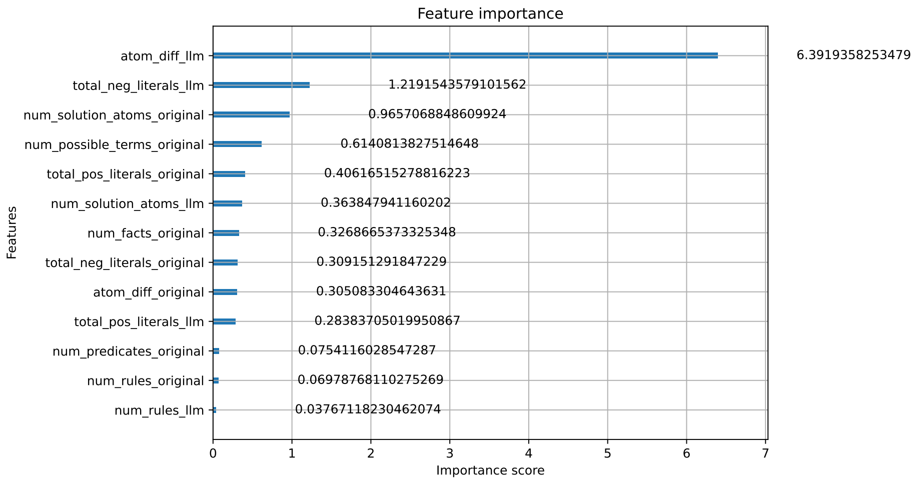

Fig. 4: XGBoost Feature Importance

5 Related Work

Recent work has explored the integration of LLMs and ASP with several approaches studying how LLMs can learn from or reason over ASP directly. Borroto et al. (2025) [1] investigates question answering and inductive learning from answer sets, demonstrating how stable models can serve as supervision signals for LLMs. Declarative knowledge distillation techniques have also been proposed to transfer structured reasoning from LLMs into symbolic representations by Either et al. (2024) [5]. Similarly, Borroto et al. (2024) [2] demonstrates automatic composition of ASP programs from natural language specifications, focusing on translating high level descriptions into executable ASP encodings.

Beyond ASP specific research, logic programming has been used as a tool to evaluate and scaffold LLM reasoning. Hybrid GOFAI LLM systems show how expert systems can be rebuilt using LLM generated logic rules by Garrido et al. (2025) [8], while Xu et al. (2025) [23] shows programming based test evaluators systematically assess LLM reasoning reliability . These works support the use of formal solvers, as in LAMP, to ground and validate LLM outputs.

Planning provides another closely related line of research. Logical chain of thought tuning for symbolic planning by Verma et al. (2025) [21] and LLM symbolic planning pipelines without expert input by Huang et al. (2024) [10] highlight the benefits of keeping solvers such as ASP planners in the loop. Established

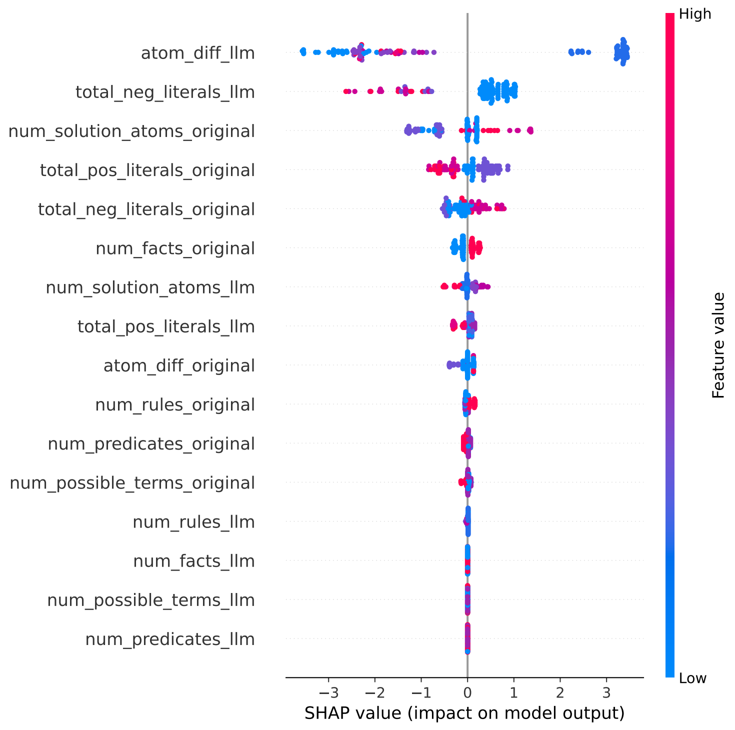

Fig. 5: SHAP Feature Importance

ASP planning benchmarks, including Plasp developed by Dimopoulos et al. [4], offer realistic domains that could extend LAMP beyond product configuration. Neuro symbolic refinement frameworks such as PEIRCE further demonstrate how LLMs and formal reasoning systems can iteratively improve each other [18] by Quan et al. (2025).

6 Future Work

Several directions for future work follow naturally. First, iterative prompting could be improved by more explicit chain of thought reasoning, potentially evaluating candidate rules across multiple instance fact sets per iteration, with Clingo acting as a tool within an LLM based agent loop. Prompt optimization frameworks such as DSPy could be used to automatically refine prompts and feedback strategies. Second, the synthetic datasets could be extended to richer ASP constructs, including higher arity predicates, choice rules, disjunction, and head disjunction, as well as more complex dependency structures such as those proposed by Gandarela et al. (2025) regarding logic program expressivity [7].

Third, evaluation could move beyond synthetic configuration tasks to existing ASP datasets, including OOASP models, real world ASP encodings, planning benchmarks such as Plasp and PDDL to ASP translations, and domains like rail scheduling. Fourth, deeper natural language interfaces to ASP could be explored, covering bidirectional translation between text and ground facts, rules, stable models, and multiple natural languages. Finally, optimization remains a promising avenue, including other ways of transforming ASP into more efficient ASP, compiling ASP fragments to lower level code, such as C, and tighter integration with solver level optimizations such as those seen in the Non Ground Optimizer (NGO), enabling LLMs to reason not only about correctness but also about grounding and solving performance.

7 Conclusion

This work proposed a structured LLM to ASP Modeling Protocol (LAMP) designed to support the deduction and optimization of ASP rules from instance facts and expected stable models. The results show this as effective for guiding LLMs toward correct and often optimized ASP encodings. By constraining the modeling task and providing iterative feedback via an ASP solver, this system reduces the modeling burden associated with ASP while preserving correctness.

The tested models exhibited clear performance differences, with *gpt-5-mini* and *gpt-oss:20b* achieving solid stable model reconstruction, however *qwen3-coder:30b* struggled despite its programming focus. These results highlight that strong general code generation ability does not necessarily translate to proficiency in logic programming, underscoring the value of ASP specific evaluation frameworks. Analysis of complexity features and feature importance, shows that the presence of negative literals and the ability to correctly infer new atoms are the strongest predictors of LLM success.

Beyond assisting users in ASP modeling, this protocol provides a system for probing LLM reasoning and logic programming capabilities. Future work includes extending to richer ASP constructs such as disjunction and choice rules, incorporating solver-level performance metrics, and exploring fine-tuned or neuro-symbolic hybrids to further improve robustness and scalability.

References

1. Borroto, M., Gallagher, K., Ielo, A., Kareem, I., Ricca, F., Russo, A.: Question answering with llms and learning from answer sets (2025), https://arxiv.org/abs/2509.16590
2. Borroto, M., Kareem, I., Ricca, F.: Towards automatic composition of asp programs from natural language specifications (2024), https://arxiv.org/abs/2403.04541
3. Coppolillo, E., Calimeri, F., Manco, G., Perri, S., Ricca, F.: Llasp: Fine-tuning large language models for answer set programming (2024), https://arxiv.org/abs/2407.18723
4. DIMOPOULOS, Y., GEBSER, M., LÜHNE, P., ROMERO, J., SCHAUB, T.: plasp 3: Towards effective asp planning. Theory and Practice of Logic Programming **19**(3), 477–504 (Jan 2019). https://doi.org/10.1017/s1471068418000583, http://dx.doi.org/10.1017/S1471068418000583

5. Eiter, T., Hadl, J., Higuera, N., Oetsch, J.: Declarative knowledge distillation from large language models for visual question answering datasets (2024), https://arxiv.org/abs/2410.09428
6. Falkner, A., Ryabokon, A., Schenner, G., Shchekotykhin, K.: Ooasp: Connecting object-oriented and logic programming (2015), https://arxiv.org/abs/1508.03032
7. Gandarela, J.P., Carvalho, D.S., Freitas, A.: Inductive learning of logical theories with llms: An expressivity-graded analysis (2025), https://arxiv.org/abs/2408.16779
8. Garrido-Merchán, E.C., Puente, C.: Gofai meets generative ai: Development of expert systems by means of large language models (202 5), https://arxiv.org/abs/2507.13550
9. Hecher, M., Kiesel, R.: Extended version of: On the structural hardness of answer set programming: Can structure efficiently confine the power of disjunctions? (2024), https://arxiv.org/abs/2402.03539
10. Huang, S., Lipovetzky, N., Cohn, T.: Planning in the dark: Llm-symbolic planning pipeline without experts (2024), https://arxiv.org/abs/2409.15915
11. Ishay, A., Yang, Z., Lee, J.: Leveraging large language models to generate answer set programs (2023), https://arxiv.org/abs/2307.07699
12. Kalyanpur, A., Saravanakumar, K.K., Barres, V., Chu-Carroll, J., Melville, D., Ferrucci, D.: Llm-arc: Enhancing llms with an automated reasoning critic (2024), https://arxiv.org/abs/2406.17663
13. Kaur, N., McPheat, L., Russo, A., Cohn, A.G., Madhyastha, P.: An empirical study of conformal prediction in llm with asp scaffolds for robust reasoning (2025), https://arxiv.org/abs/2503.05439
14. Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., Vardhamanan, S., Haq, S., Sharma, A., Joshi, T.T., Moazam, H., Miller, H., Zaharia, M., Potts, C.: Dspy: Compiling declarative language model calls into self-improving pipelines (2023), https://arxiv.org/abs/2310.03714
15. Li, N., Liu, P., Liu, Z., Dai, T., Jiang, Y., Xia, S.T.: Logic-of-thought: Empowering large language models with logic programs for solving puzzles in natural language (2025), https://arxiv.org/abs/2505.16114
16. Lin, X., Wu, Y., Yang, H., Zhang, Y., Zhang, Y., Ji, J.: Clmasp: Coupling large language models with answer set programming for robotic task planning (2024), https://arxiv.org/abs/2406.03367
17. Lundberg, S.M., Erion, G.G., Lee, S.I.: Consistent individualized feature attribution for tree ensembles (2019), https://arxiv.org/abs/1802.03888
18. Quan, X., Valentino, M., Carvalho, D.S., Dalal, D., Freitas, A.: Peirce: Unifying material and formal reasoning via llm-driven neuro-symbolic refinement (2025), https://arxiv.org/abs/2504.04110
19. Rajasekharan, A., Zeng, Y., Padalkar, P., Gupta, G.: Reliable natural language understanding with large language models and answer set programming. Electronic Proceedings in Theoretical Computer Science **385**, 274–287 (Sep 2023). https://doi.org/10.4204/eptcs.385.27, http://dx.doi.org/10.4204/EPTCS.385.27
20. Ren, L., Xiao, G., Qi, G., Geng, Y., Xue, H.: Can llms solve asp problems? insights from a benchmarking study (extended version) (2025), https://arxiv.org/abs/2507.19749
21. Verma, P., La, N., Favier, A., Mishra, S., Shah, J.A.: Teaching llms to plan: Logical chain-of-thought instruction tuning for symbolic planning (2025), https://arxiv.org/abs/2509.13351
22. Wang, R., Sun, K., Kuhn, J.: Dspy-based neural-symbolic pipeline to enhance spatial reasoning in llms (2024), https://arxiv.org/abs/2411.18564

23. Xu, Z., Ding, J., Lou, Y., Zhang, K., Gong, D., Li, Y.: Socrates or smartypants: Testing logic reasoning capabilities of large language models with logic programming-based test oracles (2025), https://arxiv.org/abs/2504.12312
24. Zeng, Y., Gupta, G.: Reliable collaborative conversational agent system based on llms and answer set programming (2025), https://arxiv.org/abs/2505.06438
25. ZENG, Y., RAJASEKHARAN, A., BASU, K., WANG, H., ARIAS, J., GUPTA, G.: A reliable common-sense reasoning socialbot built using llms and goal-directed asp. Theory and Practice of Logic Programming **24**(4), 606–627 (Jul 2024). https://doi.org/10.1017/s147106842400022x, http://dx.doi.org/10.1017/S147106842400022X