LLMASP: A Framework for Mitigating Hallucinations and Supporting Symbolic Reasoning in Large Language Models

Lorenzo Grillo[1]

[1]University of Calabria, Rende (CS) Italy

Abstract

LLMASP is a framework that integrates Large Language Models (LLMs) with Answer Set Programming (ASP) to combine the strengths of natural language understanding and formal logical reasoning. It enables the extraction of structured, domain-specific facts from natural language input using YAML-based specifications and constrained output formats defined by GBNF grammars. These facts are then fed into an ASP solver to perform precise reasoning, and the results are translated back into natural language by the LLM to improve interpretability. Currently, the proposal evaluates the performance and accuracy of different GBNF grammars by analyzing the quality of the ASP facts generated from a user input. The evaluation is done on a set of ASP problems and shows that JSON-formatted outputs achieve higher extraction accuracy, except for longer inference times, whereas CSV outputs offer faster performance with a slight reduction in precision.

Keywords

Natural Language processing, Answer Set Programming, Large Language Models, Mitigating hallucinations and Supporting reasoning, Neural Symbolic reasoning

1. Introduction and problem description

Large Language Models (LLMs) and Answer Set Programming (ASP) represent two distinct yet complementary paradigms in Artificial Intelligence. LLMs, such as GPT [1], PaLM [2], and LLaMa [3], have transformed natural language processing (NLP) by achieving unprecedented levels of fluency and understanding in textual data. Their rise was largely stimulated by the introduction of the Transformer architecture, as described in the seminal paper "Attention is All You Need" by Vaswani et al. (2017) [4], which introduced the self-attention mechanism as a powerful alternative to recurrent structures in sequence modeling. Since then, attention mechanisms have evolved significantly. Variants such as Active-Dormant Attention [5], Flash Attention [6], and ALiBi (Attention with Linear Biases) [7] have been introduced to enhance efficiency, scalability, and context preservation. In addition to architectural innovations, reasoning capabilities of LLMs have been improved through techniques such as Chain-of-Thought (CoT) prompting [8], which enables models to break down complex reasoning tasks into intermediate steps, significantly enhancing performance on tasks requiring multi-step logic. Moreover, LLMs can be guided to produce structured outputs [9] by adhering to formal grammars [10], such as GBNF (GGML Backus-Naur Form) specifications [11], allowing for constrained generation in applications requiring syntax compliance. In contrast, Answer Set Programming (ASP) provides a declarative paradigm for knowledge representation and reasoning. ASP enables systems to model complex domains using logical rules, infer consequences, and solve combinatorial problems with high efficiency [12, 13]. It has demonstrated success in applications such as planning [14], diagnosis [15], configuration [16] and decision-making. This proposal introduces a comprehensive framework that combines Large Language Models (LLMs) and Answer Set Programming (ASP), leveraging the strengths of both paradigms to address their respective limitations [17]. It presents a method for encoding domain-specific knowledge into input prompts using a YAML-based format, allowing LLMs to generate structured relational facts that are then processed by the ASP solver for reasoning. The outcomes inferred by ASP are subsequently translated back into natural language by the LLM, enhancing the user

Joint Proceedings of the Workshops and Doctoral Consortium of the 41st International Conference on Logic Programming, September 9–13, 2025, Rende, Italy

loregrillo28@gmail.com (L. Grillo)

experience and improving the interpretability of the results [18], Figure 1.

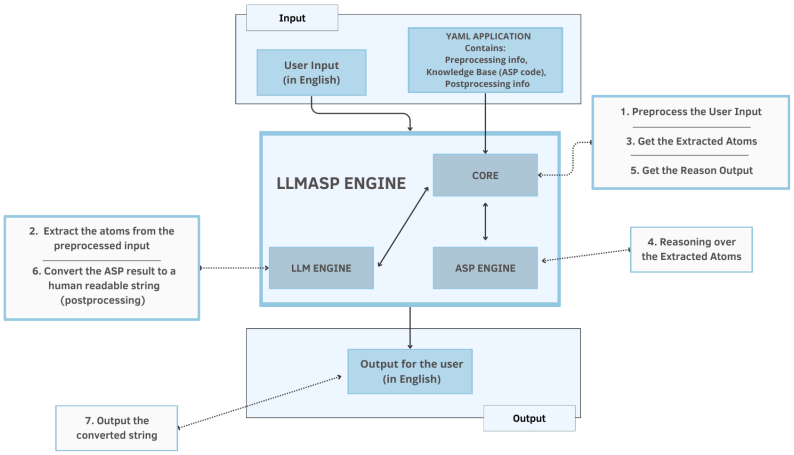


Figure 1: LLMASP Pipeline

2. Background and overview of the existing literature

Large Language Models (LLMs) are sophisticated artificial intelligence systems based on the Transformer architecture. They are trained on extensive collections of textual data to learn complex patterns and structures of human language. Given an input *prompt*, optionally accompanied by a *history* consisting of the list of previous messages, they return a *completion*, that is, a textual response that probabilistically continues or answers the input.

Moreover, LLM frameworks such as **Ollama** offer the possibility of constraining the generation process to strictly adhere to a formal grammar, specifically the Grammar Backus–Naur Form (GBNF), which can be automatically generated from a JSON structure defined using JSON Schema. By patching the Ollama framework, it is also possible to send user-defined grammars via the API, allowing for greater control over the structure of the response. This is crucial for optimizing token usage, for example by specifying output formats such as CSV. Since the GBNF grammar imposes strict constraints, it is assumed that the generated output always conforms to the expected structure.

Let's consider the following example: the task is to extract the person and the action they are performing from the user prompt. A GBNF grammar for this task is defined as follows:

```
root ::= who action
who ::= "Alice" | "Marco" | "Luca"
action ::= "eats pasta" | "plays computer games" | "sleeps"
```

The GBNF file is passed to the patched Ollama API in the following way:

```
gbnf_grammar = open("grammar.gbnf")
chat.completion(messages=["I've seen Alice eating pasta"], grammar=gbnf_grammar)
```

The output of that query will be:

```
Alice eats pasta.
```

which follows the format previously defined by the grammar.

Answer Set Programming (ASP) is a declarative, rule-based language designed for knowledge representation and reasoning, whose correctness is based on formal semantics. An ASP program consists of a collection of rules that derive new facts from a given input. It can be read by humans and the logic can be validated by domain experts. The solutions generated by an ASP program are formally referred to as answer sets, that is, collections of inferred facts that satisfy all the rules and constraints of the program.

Here's an example of an ASP program:

```
a :- not b. % means a is true if we cannot prove b.
b :- not a. % means b is true if we cannot prove a.
c :- a.     % means c is true if a is true.
```

This program has exactly two answer sets:

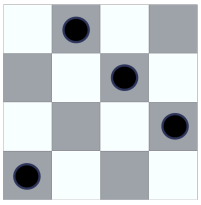
$$\{a, c\} \quad \text{and} \quad \{b\}.$$

3. Goal of the research

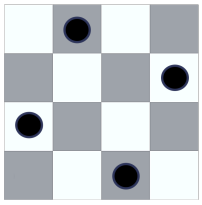
The goal of this research is to enable LLMs to perform structured reasoning by combining their strong pattern-matching abilities with the logical reasoning capabilities of Answer Set Programming (ASP). This approach allows for solving complex problems defined in ASP while also reducing hallucinations in LLM outputs [19, 20]. Since the resolution is offloaded to ASP, the problem doesn't need to be part of the training set. Users can input plain natural language, which is then translated into ASP for reasoning and resolution. To achieve this, the research will focus on the following sub-goals:

- **Conversion back to natural language**: Methods to translate the ASP result back to natural language by using a different behavior YAML. The translation makes sure that all the ASP facts are correctly included inside the produced natural language output (Hallucination mitigation).
- **Automatic generation of the ASP code**: Taking a natural language problem description, generate the required ASP logic to solve the problem.
- **Loop LLM-ASP Feedback**: A closed loop interaction where LLM outputs are iteratively refined through ASP reasoning, enabling more accurate and controlled information extraction.
- **Neural Network Layer**: Reduces latency by eliminating repeated external calls and would allow direct control over the internal mechanisms of the Transformer, such as dynamic adjustment of token weights.

4. Current status of the research



(a) LLM result without LLMASP



(b) The ASP solution of the 4-Queens problem

Figure 2: Solutions given for the N-Queens problem

```yaml
# Application YAML file structure
preprocessing:
  - size(n) : The size of the board
knowlage_base: #ASP code
    queen(y, x) :- .....
postprocessing:
    queen(y, x) : Explain to the user ...
```

(a) Structure of the Application YAML file

```yaml
# Behavior YAML file structure
preprocessing:
    init:
    You are a Natural Language to Datalog
    ...
mapping: ...
postprocessing: ...
```

(b) Structure of the Behavior YAML file

Figure 3: YAML Files

The current state of the research involves the definition of a structured format to orchestrate the interaction between the LLM and the ASP solver. User input is treated as a textual database. Leveraging GBNF grammar and a predefined protocol, the LLM is able to query and extract relevant atoms, if present in the input, and convert them into ASP facts. These atoms are specified within a YAML configuration file, which also defines the model's behavior, including the System Prompt and concise descriptions of each atom to be extracted. The extracted facts are then passed to the ASP solver, which computes and returns the solution to the defined problem.

To illustrate the proposed approach, consider the following example. The task is to solve the well-known N-Queens problem, described as follows:

> "The N-Queens puzzle consists in placing n queens on an n×n chessboard such that no two queens threaten each other. Given an integer n, return one solution to the N-Queens puzzle."

The LLM used in this example is LLAMA 3.1 7B. When provided with the problem description and the following prompt:

> Return all the solutions for n=4.

the LLM produces the output shown in Figure 2a, which is incorrect because the queens threaten each other diagonally.

Now, consider the same LLM combined with the LLMASP framework. LLMASP requires an Application YAML file containing the problem statement and the definition of the facts to be extracted 3a, as well as a behavior file specifying how the LLM should extract the data and which intermediate format should be used 3b. The intermediate format can be either CSV or JSON, and the format specification is used to control the output structure of the LLM. When CSV is selected, the LLM produces a simple, tabular output such as:

```
head \t body \n
...
```

which contains only the essential components required to construct an ASP fact, separated by tab separator. When JSON is selected, the LLM follows the rules defined by the GBNF grammar automatically generated by Ollama. The output is structured as:

```
{ "list_fact1": [{"head" : "body"}, ] }
```

Using the same prompt, the LLM extracts the following fact:

```
size(4).
```

which is subsequently passed to the ASP solver, returning:

```
queen(1, 2). queen(2, 4).  queen(3, 1). queen(4, 3).
```

This solution is correct, as the queens do not threaten each other, as shown in figure 2b.

Problem	JSON (PD)		JSON		CSV		CSV (PD)	
	ACC	F1	ACC	F1	ACC	F1	ACC	F1
Incremental Scheduling	0.449	0.620	0.445	0.616	0.282	0.440	0.381	0.552
Graph Colouring	0.636	0.778	0.869	0.930	0.860	0.925	0.830	0.907
Graceful Graphs	0.938	0.968	0.875	0.933	0.719	0.836	0.781	0.877
Crossing Minimization	0.927	0.962	0.918	0.957	0.864	0.927	0.932	0.965
Connected Maximum-density Still Life	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Valves Location Problem	0.782	0.877	0.694	0.819	0.782	0.877	0.807	0.893
Stable Marriage	0.550	0.710	0.667	0.800	0.659	0.794	0.604	0.753
Sokoban	0.776	0.874	0.809	0.894	0.648	0.787	0.659	0.795
Ricochet Robots	0.485	0.653	0.442	0.613	0.333	0.500	0.382	0.553
Permutation Pattern Matching	0.567	0.723	0.439	0.610	0.269	0.424	0.251	0.402
Partner Units	1.000	1.000	1.000	1.000	0.959	0.979	1.000	1.000
Nomystery	0.803	0.891	0.766	0.868	0.538	0.700	0.543	0.704
Maximal Clique Problem	0.812	0.897	0.963	0.981	0.951	0.975	0.872	0.932
Labyrinth	0.973	0.986	0.964	0.982	0.987	0.994	0.942	0.970
Knight Tour With Holes	1.000	1.000	1.000	1.000	0.806	0.892	1.000	1.000
Knight Tour	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000

Table 1
Accuracy (ACC) and F1 scores of each configuration tested

5. Preliminary results accomplished

In order to verify the correct extraction of the information encoded in ASP, several tests have been conducted to determine the most suitable setup for this component. The tests were conducted on the following hardware configuration: an RTX 3070 GPU, an Intel i7-11700KF CPU, and 32GB of DDR4 RAM. They focused on evaluating the precision of ASP fact extraction from user input. It also measured performance and examined which additional information, in this case the problem description, could improve the accuracy of the LLM's extraction.

The execution time of each method (JSON and CSV) is evaluated by repeating the tests 10 times to verify consistency. The language model produced identical outputs across all runs, as it was configured with a temperature value of 0 and a top_p parameter set to 0.98.

These parameters are commonly used to regulate the stochasticity of the model's responses. Specifically, the temperature parameter influences the degree of creativity in the output: higher values lead to more diverse and imaginative responses, while lower values favor determinism and repeatability[21]. Similarly, the top_p parameter limits the sampling space to the top p most probable tokens, with lower values introducing greater variability and higher values promoting more stable and predictable results. The precision of the extracted facts was evaluated using standard classification metrics, including true positives, true negatives, false positives, and false negatives, from which the F1-Score was derived. In the initial test configuration, the problem description was included in the system prompt to provide additional context to the language model.

The results shown in the table 1, indicate that the JSON format with the problem description inside the system prompt (PD) provided an higher level of precision compared to the CSV (PD) format also with the same setup. However, this improvement in accuracy was accompanied by an increase in execution time. Specifically, the JSON GBNF grammar required an average of 665 seconds (approximately 11 minutes) to complete the tests, whereas the CSV GBNF grammar approach completed in an average of 540 seconds (approximately 9 minutes). In the second test, the problem description was omitted from the system prompt. The JSON format maintained approximately the same level of precision. In some cases, performance slightly decreased, while in others it improved. Similarly, the CSV format exhibited a decline in performance for certain problems but demonstrated improved precision in others.
As can be observed in table 2, the JSON-based method produces approximately 100,000 more tokens

JSON	505,084
CSV	391,797

Table 2
Total number of tokens generated in each mode.

than the CSV-based method. This significant difference reflects the greater verbosity and structural complexity typically associated with the JSON format. While the increased token count may contribute to slightly higher precision by providing the language model with more explicit and structured information, it also leads to longer processing times and greater computational cost.
On the other hand, the CSV format, being more concise and lightweight, results in faster response times and reduced token usage. However, this compactness may occasionally limit the model's ability to fully interpret the intended meaning, potentially leading to a loss in extraction precision.

6. Open issues and expected achievements

The proposed framework still has several unresolved issues. Notably, the need to invoke the LLM once per atom significantly slows down the extraction process compared to a single-pass LLM inference. Additionally, the LLM may occasionally enter an infinite generation loop, repeatedly producing the same token. This behavior has been partially mitigated by using the streaming mode of the Ollama API to detect token repetition: when a loop is identified, the chat.completion function is forcibly stopped, and the system returns the partial output extracted up to that point. When the information extracted is not sufficient to solve the problem, an Active-Prompting system can be employed to request additional input in order to satisfy the requirements.
The expected achievements include reaching logic problem solving performance comparable to larger models, showing that complex reasoning can be done efficiently also with the smaller ones. Additionally, the framework can be designed to run on low powered devices, enabling embedded hardware to handle complex tasks without relying on powerful hardware.

Declaration on Generative AI

During the preparation of this work, the author used ChatGPT, Grammarly in order to: Grammar and spelling check, Paraphrase and reword. After using this tool/service, the author reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] T. B. Brown, et al., Language models are few-shot learners, CoRR abs/2005.14165 (2020). URL: https://arxiv.org/abs/2005.14165. arXiv:2005.14165.

[2] A. Chowdhery, et al., Palm: Scaling language modeling with pathways, J. Mach. Learn. Res. 24 (2023) 240:1–240:113. URL: http://jmlr.org/papers/v24/22-1144.html.

[3] H. Touvron, et al., Llama: Open and efficient foundation language models, CoRR abs/2302.13971 (2023). doi:10.48550/ARXIV.2302.13971. arXiv:2302.13971.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, https://arxiv.org/abs/1706.03762, 2023. arXiv:1706.03762.

[5] T. Guo, D. Pai, Y. Bai, J. Jiao, M. I. Jordan, S. Mei, Active-dormant attention heads: Mechanistically demystifying extreme-token phenomena in llms, https://arxiv.org/pdf/2410.13835, 2024. arXiv:2410.13835.

[6] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, C. Ré, Flashattention: Fast and memory-efficient exact attention with io-awareness, https://arxiv.org/pdf/2205.14135, 2022. arXiv:2205.14135.

[7] O. Press, N. A. Smith, M. Lewis, Train short, test long: Attention with linear biases enables input length extrapolation, https://arxiv.org/abs/2108.12409, 2022. arXiv:2108.12409.

[8] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, D. Zhou, Chain-of-thought prompting elicits reasoning in large language models, https://arxiv.org/pdf/2201.11903, 2023. arXiv:2201.11903.

[9] M. X. Liu, F. Liu, A. J. Fiannaca, T. Koo, L. Dixon, M. Terry, C. J. Cai, "we need structured output": Towards user-centered constraints on large language model output, in: Extended Abstracts of the CHI Conference on Human Factors in Computing Systems, CHI EA '24, Association for Computing Machinery, New York, NY, USA, 2024. URL: https://doi.org/10.1145/3613905.3650756. doi:10.1145/3613905.3650756.

[10] D. Deutsch, S. Upadhyay, D. Roth, A general-purpose algorithm for constrained sequential inference, in: M. Bansal, A. Villavicencio (Eds.), Proceedings of the 23rd Conference on Computational Natural Language Learning, CoNLL 2019, Hong Kong, China, November 3-4, 2019, Association for Computational Linguistics, 2019, pp. 482–492. URL: https://doi.org/10.18653/v1/K19-1045. doi:10.18653/V1/K19-1045.

[11] D. D. McCracken, E. D. Reilly, Backus-Naur form (BNF), John Wiley and Sons Ltd., GBR, 2003, p. 129–131.

[12] V. Marek, M. Truszczyński, Stable models and an alternative logic programming paradigm, in: The Logic Programming Paradigm: a 25-year Perspective, 1999, pp. 375–398. doi:10.1007/978-3-642-60085-2_17.

[13] I. Niemelä, Logic programming with stable model semantics as a constraint programming paradigm, Annals of Mathematics and Artificial Intelligence 25 (1999) 241–273. doi:10.1023/A:1018930122475.

[14] P. Cappanera, M. Gavanelli, M. Nonato, M. Roma, Logic-based benders decomposition in answer set programming for chronic outpatients scheduling, TPLP 23 (2023) 848–864. doi:10.1017/S147106842300025X.

[15] F. Wotawa, On the use of answer set programming for model-based diagnosis, in: H. Fujita, P. Fournier-Viger, M. Ali, J. Sasaki (Eds.), IEA/AIE 2020, Kitakyushu, Japan, September 22-25, 2020, Proceedings, volume 12144 of *LNCS*, Springer, 2020, pp. 518–529. doi:10.1007/978-3-030-55789-8_45.

[16] R. Taupe, G. Friedrich, K. Schekotihin, A. Weinzierl, Solving configuration problems with ASP and declarative domain specific heuristics, in: M. Aldanondo, A. A. Falkner, A. Felfernig, M. Stettinger (Eds.), Proceedings of the 23rd International Configuration Workshop (CWS/ConfWS 2021), Vienna, Austria, 16-17 September, 2021, volume 2945 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 13–20. URL: https://ceur-ws.org/Vol-2945/21-RT-ConfWS21_paper_4.pdf.

[17] S. Williams, J. Huckle, Easy problems that llms get wrong, 2024. URL: https://arxiv.org/abs/2405.19616. arXiv:2405.19616.

[18] M. Alviano, L. Grillo, F. Lo Scudo, L. A. R. Reiners, Integrating answer set programming and large language models for enhanced structured representation of complex knowledge in natural language, in: IJCAI 2025, Montreal, Canada, August 16-22, 2025, https://tinyurl.com/ijcai25-llmasp, 2025.

[19] Y. Bang, Z. Ji, A. Schelten, A. Hartshorn, T. Fowler, C. Zhang, N. Cancedda, P. Fung, Hallulens: Llm hallucination benchmark, 2025. URL: https://arxiv.org/abs/2504.17550. arXiv:2504.17550.

[20] G. Perković, A. Drobnjak, I. Botički, Hallucinations in llms: Understanding and addressing challenges, in: 2024 47th MIPRO ICT and Electronics Convention (MIPRO), 2024, pp. 2084–2088. doi:10.1109/MIPRO60963.2024.10569238.

[21] L. Li, L. Sleem, N. Gentile, G. Nichil, R. State, Exploring the impact of temperature on large language models:hot or cold?, 2025. URL: https://arxiv.org/abs/2506.07295. arXiv:2506.07295.