Lightweight LLMs for Encoding ASP

Discussion paper

Irfan Kareem[1], Manuel Borroto Santana[1], Francesco Ricca[1] and Michele Vitale[1,*]

[1]University of Calabria, Rende, Italy

Abstract

Translating natural-language specifications into Answer Set Programming (ASP) remains a complex task, especially with respect to semantic fidelity and robust reasoning. Recent approaches increasingly rely on large, resource-intensive language models, leaving the capabilities of smaller and locally deployable models comparatively underexplored. In recently accepted work, we extended prior research on NL2ASP by investigating whether fine-tuned, lightweight, open-source language models can effectively translate natural language into ASP. We introduced a more natural and linguistically diverse dataset and evaluated models from the Qwen, Llama, and Ministral families using both a Controlled Natural Language (CNL) intermediate representation and direct ASP generation. The results show that fine-tuned lightweight LLM models outperform the earlier transformer-based NL2ASP approach and can offer a practical alternative to substantially larger models. This discussion paper summarises those findings, examines their implications and limitations, and outlines directions for improving semantic robustness and integrating lightweight models into agentic pipelines.
This work has already been accepted at the 18th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2026).

Keywords

Answer Set Programming, Large Language Models, Natural Language Processing, Controlled Natural Language

1. Introduction

Large Language Models (LLMs) have changed software development workflows by supporting the generation, completion, and revision of source code [1, 2]. Knowledge representation is a particularly interesting application of this capability: automatically translating an informal problem description into a formal specification could make declarative systems accessible to users who do not know their syntax. In this paper, we focus on translating natural language (NL) into Answer Set Programming (ASP), a declarative paradigm in which the properties of solutions are represented by logical rules [3]. The task therefore goes beyond generating text that merely resembles code, since the output must recover the intended ASP structure and preserve the meaning of the input specification in the resulting rules.

Earlier work introduced NL2ASP, a two-stage architecture that translates NL into an ASP-oriented Controlled Natural Language (CNL) and then deterministically compiles CNL into ASP [4, 5]. Other studies have fine-tuned language models for recurring ASP patterns [6] or used large models in multi-step, tool-supported pipelines [7, 8]. These results are encouraging, but the most capable agentic systems commonly depend on large, closed-source or reasoning-based models. Such dependence increases inference cost and often limits local deployment, particularly where privacy, reproducibility, or hardware constraints matter. Our recently accepted study [9] investigated whether open-weight LLMs with fewer than ten billion parameters can provide an effective alternative. We revisited the NL2ASP architecture using fine-tuned lightweight models, redesigned its dataset to provide more natural and varied inputs, and compared CNL-mediated translation with direct ASP generation. We also studied the contribution of fine-tuning, predicate information, and agentic execution. This discussion paper summarises that

AIxIA 2026 Discussion Papers, 24th International Conference of the Italian Association for Artificial Intelligence, Perugia, Italy, 2026

*Corresponding author.

irfan.kareem@unical.it (I. Kareem); manuel.borroto@unical.it (M. B. Santana); francesco.ricca@unical.it (F. Ricca); michele.vitale@unical.it (M. Vitale)

0000-0002-9762-3954 (I. Kareem); 0000-0002-9546-1998 (M. B. Santana); 0000-0001-8218-3178 (F. Ricca); 0009-0008-5853-7560 (M. Vitale)

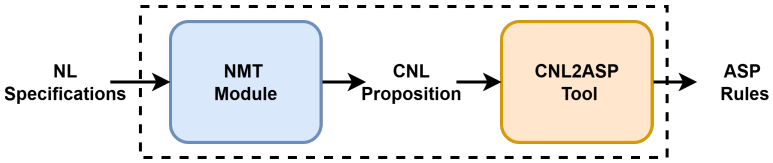

Figure 1: NL2ASP two-step architecture for automatic composition of ASP programs [9].

study and then reflects on what its results imply for reliable and resource-conscious NL-to-ASP systems. In particular, we examine the limitations of the approach and outline directions for improving semantic robustness and integrating lightweight models into agentic pipelines.

2. Overview

Translation pipelines The study evaluates two pipelines for translating natural-language statements into ASP. The first follows the NL2ASP architecture (see Figure1), in which a neural machine translation (NMT) module tanslates a natural-language specification into Controlled Natural Language (CNL) [5], which serves as an intermediate representation. In this setting, the original transformer-based translation module is replaced with a fine-tuned lightweight LLM that maps natural language to CNL. Then, the CNL2ASP [5] tool deterministically translates the CNL output into ASP. The second pipeline bypasses the intermediate representation and uses a fine-tuned lightweight LLM to generate ASP directly from natural language. This direct approach simplifies the architecture and avoids CNL-induced failures, but places full responsibility for syntactic and semantic validity on the model.

Dataset creation To fine-tune the models, we address some limitations of the original NL2ASP corpus. The corpus was strongly influenced by mathematical phrasing and explicit variables, which are uncommon in ordinary user descriptions. Moreover, its template-based construction methodology required predefined patterns that introduce linguistic redundancy, and constrain the resulting language. To overcome these limitations, we replaced both the original dataset and its construction methodology by building a new corpus. We started from 534 representative ASP rules collected from graph-related problems in competitions, books, the NL2ASP repository, and well-established benchmark problems. Using ASP2CNL, we manually crafted corresponding CNL sentences and paired them with multiple manual NL descriptions, and the corresponding CNL category, yielding an initial set of 1,310 tuples of the form $\langle NL, CNL, ASP, Category \rangle$.

To augment this data without introducing redundancy, we selected one distinctive NL description per rule and applied an LLM-driven, manually verified synonym-substitution pipeline. This stage produced 2,136 examples. A final rephrasing step further diversified the NL inputs, generating three variants per instance, except for the frequent Definition Whenever category, which was rephrased once. After merging these variants and including the descriptions that were initially set aside, the final training set contained 7,151 instances. Dataset quality was assessed through manual inspection and LLM-as-a-judge evaluation, that processed each tuple against its ASP rule; low-scoring cases were manually corrected. For testing, we manually constructed a set of 200 instances distributed across the seven rule categories. To this end, we selected a subset of the rules used to build the training set and rewrote their natural-language descriptions using different linguistic formulations. We also included rules from graph-related problems not considered during training. Finally, a predicate-information-enriched version of the test set was created as well.

3. Summary of Results

The experiments considered LLM models Qwen3 8B, Qwen3.5 9B, Llama3.1 8B, and Ministral 8B. Each model was fine-tuned using Low-Rank Adaptation (LoRA) with ranks 8, 16, and 32 [10]. Encoder–decoder baselines, i.e., T5-small, T5-large, and T5-3B, were fine-tuned using standard supervised fine-tuning.

Evaluation distinguished *syntactic accuracy*, checked with CNL2ASP for CNL output and Clingo for ASP outputs, from *semantic accuracy*, which was independently assessed by three human evaluators. A sentence was considered semantically correct only when the generated ASP rule preserved the meaning of the reference. Disagreements among the evaluators were resolved through dedicated discussions among the experts. The experiments first selected the best LoRA configuration within each model family and compared it with the T5 baselines. They then examined direct NL-to-ASP generation, zero-shot use, and the substitution of compact models into an existing agentic pipeline.

Fine-tuning and model selection All decoder-only models achieved high syntactic accuracy when generating CNL. The best syntactic result was obtained by Ministral 8B with LoRA rank 32, reaching 99.0%, while the best semantic result was obtained by Qwen3.5 9B with rank 16, reaching 82.0% (Table 1). This difference shows that compilation success alone is not a reliable criterion for model selection. For instance, Ministral 8B at rank 32 had the highest syntactic score but only 55.0% semantic accuracy. Looking beyond the aggregate scores, the strongest Qwen3.5 9B configuration performed well on definitions and strong constraints, but remained weaker on quantified choice rules, which reached only 46.67% semantic accuracy. T5-large reached 97.0% syntactic accuracy but only 60.0% semantic accuracy. Scaling the earlier encoder-decoder architecture thus improved grammar conformance, but many syntactically valid outputs still failed to preserve the intended meaning.

CNL mediation versus direct NL-to-ASP generation Direct generation was competitive with the CNL-mediated pipeline. Ministral 8B obtained 97.5% syntactic and 78.5% semantic accuracy, Llama3.1 8B matched the semantic score with 96.0% syntactic accuracy, and Qwen3.5 9B obtained 95.0% and 78.0%. The best CNL-mediated result (Qwen3.5 9B 82%) therefore retained only a 3.5% semantic advantage over the best direct NL-to-ASP generation. This demonstrates that the direct translation offered a simpler, single-stage pipeline at a modest accuracy cost. The baseline model T5-large was much less successful without CNL mediation, falling to 70.5% syntactic and 46.0% semantic accuracy from 97.0%/60.0% with CNL. This contrast suggests that modern decoder-only models absorb more of the target grammar directly during fine-tuning, whereas the older encoder–decoder baseline benefits more strongly from the intermediate controlled natural language.

Fine-tuning and agentic execution To assess the role of fine-tuning, the study also evaluated non-fine-tuned models on the same test set. The best zero-shot result was obtained by the reasoning-based Qwen3.5 9B variant, with 64.0% syntactic and 47.0% semantic accuracy; the non-reasoning variant and the other lightweight models performed worse. These scores remain far below the best fine-tuned configuration, which reached 98.0% syntactic and 82.0% semantic accuracy, confirming that reasoning helps but task-specific fine-tuning is crucial for reliable NL-to-ASP translation. We also evaluated lightweight models within the ASP-Bench agentic approach [8], replacing the large models used in the original setting with compact, non-fine-tuned LLMs. Qwen3.5 9B solved Maximal Clique and Maximum

Table 1
Selected results (%) on the 200-instance test set, from [9]. Best in each group in bold; overall best underlined.

Setting	Model	Syntax	Semantics
CNL-mediated, fine-tuned	Qwen3.5 9B ($r = 16$)	98.0	**<u>82.0</u>**
CNL-mediated, fine-tuned	Qwen3 8B ($r = 16$)	94.0	75.0
CNL-mediated, fine-tuned	Llama3.1 8B ($r = 8$)	95.5	69.5
CNL-mediated, fine-tuned	Ministral 8B ($r = 32$)	**<u>99.0</u>**	55.0
CNL-mediated baseline	T5-large	97.0	60.0
Direct NL-to-ASP, fine-tuned	Ministral 8B	**97.5**	**78.5**
Direct NL-to-ASP, fine-tuned	Llama3.1 8B	96.0	**78.5**
Direct NL-to-ASP, fine-tuned	Qwen3.5 9B	95.0	78.0
Direct NL-to-ASP baseline	T5-large	70.5	46.0
Direct zero-shot reasoning	Qwen3.5 9B	**64.0**	**47.0**
Direct zero-shot	Qwen3.5 9B	59.0	27.5
Direct zero-shot	Llama3.1 8B	54.0	19.5
Direct zero-shot	Ministral 8B	46.0	15.0

Independent Set, but failed on Travelling Salesperson and timed out on Connected Dominating Set; Qwen3 8B solved only Maximum Independent Set. Llama3.1 8B and Ministral 8B produced few correct rules and did not exploit tool feedback. None of the lightweight models solved two additional easy ASP-Bench instances, whereas a large commercial model solved all problems in the comparison. These results suggest that agentic orchestration alone does not compensate for limited task knowledge or unreliable tool use, reinforcing the need for task-specific adaptation.

4. Discussion and Future Research Directions

The results summarised above indicate that lightweight open-weight LLMs can be effective for NL-to-ASP translation when they are adapted to the task. The strongest evidence comes from the gap between fine-tuned and zero-shot configurations: compact models can generate useful ASP-oriented outputs, but their reliability depends heavily on task-specific training data. At the same time, the experiments also show that syntactic correctness is not sufficient. Several configurations produced outputs that were accepted by the relevant grammar or ASP parser, while still failing to preserve the intended meaning of the input specification. Semantic robustness is therefore the central challenge for practical deployment. In this sense, the goal is not only to generate valid-looking ASP code, but to ensure that the generated rules faithfully encode the intended problem constraints and solution properties.

The comparison between CNL-mediated and direct generation further suggests that the two architectures should be understood as complementary design choices. Direct NL-to-ASP generation simplifies the pipeline and avoids errors introduced by malformed CNL output. Its competitive performance shows that fine-tuned lightweight models can learn a substantial part of the target formalism directly. Nevertheless, the CNL-mediated pipeline still obtained the best semantic result and provides an intermediate representation that is easier to inspect, validate, and correct. This can be particularly valuable in settings where transparency and error localisation are important. A practical NL-to-ASP system may therefore combine both strategies, using direct generation when confidence is high and relying on CNL when additional structure or user-facing explanations are needed.

The current approach also has some limitations. The most significant one is that it operates at the level of isolated natural-language statements. In practice, the input is treated as a bag of rule-level statements, each translated independently into ASP. This approach is useful for isolating translation errors, but it does not yet address the more demanding task of deriving a complete ASP program from a full natural-language problem description. Full specifications require decomposition into simpler rule statements, stable predicate choices across those statements, and global consistency checks on the resulting program. Beyond this, the dataset still contains mainly graph-based problems, and semantic correctness was manually assessed. Manual evaluation provides reliable judgments, but it is expensive and difficult to scale to larger benchmarks and broader domains.

These limitations define the next research steps. First, the dataset should be extended beyond graph-oriented examples and should include complete problem descriptions paired with decomposed rule-level statements and validated ASP programs. Such data would support the transition from rule translation to full program construction. Second, lightweight reasoning models should be fine-tuned not only to translate individual statements, but also to support decomposition, semantic repair, and solver-guided revision. Third, the translator can be integrated into a lightweight agentic architecture. A decomposer agent could transform a full natural-language specification into simpler NL rule statements, while a modeler agent could invoke the fine-tuned translator through a dedicated Model Context Protocol server to obtain ASP rules. The resulting program could then be validated and repaired in an iterative loop. This architecture would preserve the strength of the current system at rule-level translation, while addressing its main limitation by embedding it within a broader process for constructing complete ASP encodings. Future work may also explore emerging and more expressive semantics, such as ASPQ [11, 12, 13]. All in all, the results are encouraging. They show concrete potential in the proposed approach, while also open a broad and ambitious research path toward more general and robust solutions for encoding of ASP from natural language on “small” models.

Acknowledgments

This work was supported by the Italian Ministry of Enterprises and Made in Italy under PON RIC project ASVIN "Assistente Virtuale Intelligente di Negozio" (CUP B29J24000200005); by MOZART "Modelli e tecniche di mOdernizzazione di applicaZioni e data silos a supporto di clienti esterni, field force e impiegati di backoffice basati su AI GeneRativa e apprendimento auTomatico" (CUP J49I24001740005); by SIGENERA "Large Language Models (LLM) per il controllo di SIstemi informativi, la GENERazione automatica di testo, e l'accesso a basi di conoscenzA" (CUP J29I24001770005). Francesco Ricca is a member of Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM).

Declaration on Generative AI

Large language models were used to check spelling and grammar, and to support improvements in text quality. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] A. M. Dakhel, V. Majdinasab, A. Nikanjam, F. Khomh, M. C. Desmarais, Z. M. J. Jiang, Github copilot AI pair programmer: Asset or liability?, J. Syst. Softw. 203 (2023) 111734.

[2] E. Kalliamvakou, Research: quantifying github copilot's impact on developer productivity and happiness, The GitHub Blog (2022).

[3] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming at a glance, Commun. ACM 54 (2011) 92–103.

[4] M. A. B. Santana, I. Kareem, F. Ricca, Towards automatic composition of ASP programs from natural language specifications, in: IJCAI, 2024, pp. 6198–6206.

[5] S. Caruso, C. Dodaro, M. Maratea, M. Mochi, F. Riccio, CNL2ASP: converting controlled natural language sentences into ASP, Theory Pract. Log. Program. 24 (2024) 196–226.

[6] E. Coppolillo, F. Calimeri, G. Manco, S. Perri, F. Ricca, LLASP: fine-tuning large language models for answer set programming, in: KR, 2024.

[7] S. Szeider, Cp-agent: Agentic constraint programming, in: LLM4CODE@ICSE, ACM, 2026, pp. 6–13.

[8] S. Szeider, Asp-bench: From natural language to logic programs, in: Proceedings of the 2026 IEEE/ACM 2nd International Workshop on Neuro-Symbolic Software Engineering, NSE '26, ACM, New York, NY, USA, 2026, p. 1–8. doi:10.1145/3786168.3788402.

[9] I. Kareem, M. B. Santana, F. Ricca, M. Vitale, Lightweight language models for encoding asp from natural language, in: Proceedings of the 18th International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR 2026), Klagenfurt, Austria, 2026. Accepted for publication.

[10] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, Lora: Low-rank adaptation of large language models, in: ICLR, 2022.

[11] G. Amendola, F. Ricca, M. Truszczynski, Beyond NP: quantifying over answer sets, Theory Pract. Log. Program. 19 (2019) 705–721.

[12] G. Mazzotta, F. Ricca, M. Truszczynski, Quantifying over optimum answer sets, Theory Pract. Log. Program. 24 (2024) 716–736.

[13] D. Azzolini, G. Mazzotta, F. Ricca, Probabilistic reasoning within answer set programming with quantifiers, in: KR, 2026.