Solving MaxSAT Problems from Natural Language Descriptions with LLMs and PySAT

Pedro Orvalho
Artificial Intelligence Research Institute (IIIA), Consejo Superior de Investigaciones Científicas (CSIC), Barcelona, Catalonia, Spain

Marta Kwiatkowska
Department of Computer Science, University of Oxford, Oxford, UK

Guillem Alenyà
Institut de Robòtica i Informàtica Industrial (IRI-CSIC-UPC), Barcelona, Catalonia, Spain

Felip Manyà
Artificial Intelligence Research Institute (IIIA), Consejo Superior de Investigaciones Científicas (CSIC), Barcelona, Catalonia, Spain

Abstract

Large Language Models (LLMs) provide a flexible interface for interpreting natural language problem descriptions, but they remain unreliable when asked to solve constrained optimisation tasks directly. We study a neuro-symbolic approach in which an LLM translates a natural language description of an optimisation problem into executable Python code using `PySAT`. The generated program constructs a weighted partial Maximum Satisfiability (MaxSAT) instance, invokes a MaxSAT solver, and returns the resulting assignment in a prescribed output format. In this way, the LLM is used primarily for semantic parsing and modelling, while the optimisation step is delegated to an exact symbolic solver. We outline an end-to-end pipeline consisting of natural language input, intermediate encoding plans, `PySAT` code generation, MaxSAT solving through `RC2`, and independent validation of returned solutions. This workshop abstract distils the main idea of using LLMs as natural language front-ends for solver-backed MaxSAT modelling, with the goal of making MaxSAT technology more accessible to users who do not write formal encodings by hand.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Computing methodologies → Natural language processing

Keywords and phrases Maximum Satisfiability, Large Language Models, Natural Language Interfaces, PySAT, Neuro-symbolic AI

Digital Object Identifier 10.48550/arXiv.2605.29687

Related Version https://arxiv.org/abs/2605.29687

Acknowledgements This work was supported by grant PID2022-139835NB-C21 funded by MCIN/-AEI/10.13039/501100011033. This project has received funding from the European Union's HORIZON-MSCA-2025-PF research and innovation programme under the Marie Skłodowska-Curie (Sherlock4Py, GA No 101269051). This project was additionally supported by the ALLIES Cofund, and has received funding from the European Union's Horizon-MSCA-2022-COFUND-01 research and innovation programme under the Marie Skłodowska-Curie (GA No 101126626). This project also received funding from ELSA: European Lighthouse on Secure and Safe AI project (GA No. 101070617 under UK guarantee). PO acknowledges travel support from ELSA Mobility Program (GA No 101070617).

1 Introduction

Maximum Satisfiability (MaxSAT) [1, 8] is a natural formalism for Boolean optimisation problems with hard constraints and weighted objectives, but writing MaxSAT encodings

requires modelling expertise: users must define variables, clauses, weights, and solver calls. This limits accessibility for users who can describe a problem informally but cannot easily produce a formal encoding.

We study whether Large Language Models (LLMs) can bridge this gap by translating natural language descriptions into executable `PySAT` [6] code. Rather than asking an LLM to solve an optimisation problem internally, we use it to construct a weighted partial MaxSAT formula and call `RC2` [7] through `PySAT`. The resulting pipeline follows a simple division of labour: the LLM performs language interpretation and code generation, while the MaxSAT solver performs exact optimisation. The program returns the selected assignment in a fixed JSON format, enabling automatic validation. Our focus is not to introduce a new generic solver-in-the-loop architecture, following prior neuro-symbolic integrations of LLMs with SMT, planning, and constraint solvers [3, 4, 11, 12, 14]. Rather, we provide an empirical and methodological study of natural-language-to-MaxSAT encoding, including canonical verification of *feasibility* and *optimality* under weighted soft constraints.

2 LLM-to-MaxSAT Pipeline

Given a natural language problem description, the LLM generates Python code that encodes the instance as MaxSAT using `PySAT`. Descriptions may include objects, constraints, incompatibilities, capacities, coverage requirements, or optimisation criteria. In one variant, the model first writes an intermediate encoding plan listing variables, hard clauses, soft clauses, weights, and the output schema; the plan is used only to structure code generation.

The generated program is executed and, when successful, `RC2` returns an optimal model for the produced formula. The program then converts this model into the required output format, such as a selected set, schedule, assignment, or cover. Errors, malformed outputs, or validation failures may be fed back to the LLM for a bounded number of repair attempts. Thus, solver optimality and semantic correctness are distinct: `RC2` optimises the generated formula, while external validation checks whether that formula captures the intended natural language problem.

3 Preliminary Results

To evaluate the proposed approach, we constructed a dataset of reasoning problems with natural-language descriptions and reference MaxSAT encodings, covering three benchmark families: maximum independent set (MIS), scheduling, and set cover. The dataset contains 100 instances per family. A solution is accepted only if it is both feasible and optimal with respect to the independently verified reference encoding. Table 1, in Appendix C, reports acceptance rates for direct LLM solving, chain-of-thought (CoT), program-of-thought (PoT), and two `PySAT`-based configurations. Our results show a clear benefit from using `PySAT` as an external optimisation backend. Direct answering fails completely on MIS and scheduling for all models, while PoT remains substantially below the `PySAT`-based approaches. For example, on scheduling, Gemini-3.1-Pro [2] improves from 14% with PoT to 59% with `PySAT` and planning, and GPT-5.5 [10] improves from 9% to 56%. On MIS, Gemini-3.1-Pro improves from 11% to 56%, and GPT-5.5 from 13% to 51%. Averaged across all problems and models, acceptance is 7.8% for direct answering, 11.8% for CoT, 18.2% for PoT, 40.3% for `PySAT`, and 43.4% for `PySAT` with intermediate planning.

4 Discussion and Conclusion

Our results support using LLMs as MaxSAT modelling assistants rather than black-box optimisers. Natural language offers an accessible interface for specifying Boolean optimisation problems, while `PySAT` and `RC2` provide exact solver-backed optimisation and verifiable outputs. Moreover, the generated code creates a useful debugging boundary: failures can be traced to interpretation, encoding, or output conversion errors. Future work should combine LLM-based modelling with interactive clarification, and other repair mechanisms. Overall, the approach connects flexible natural language interfaces with reliable MaxSAT solving by delegating optimisation to symbolic solvers and using LLMs only for the translation step.

References

1 Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. Maximum satisfiability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 929–991. IOS Press, 2021. `doi:10.3233/FAIA201008`.

2 Google. Gemini-3.1-Pro `https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-1-pro/`, 2026. Released: 2026-02-19.

3 Yilun Hao, Yongchao Chen, Yang Zhang, and Chuchu Fan. Large language models can solve real-world planning rigorously with formal verification tools. In Luis Chiruzzo, Alan Ritter, and Lu Wang, editors, *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 1: Long Papers, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, pages 3434–3483. Association for Computational Linguistics, 2025. URL: `https://doi.org/10.18653/v1/2025.naacl-long.176`, `doi:10.18653/V1/2025.NAACL-LONG.176`.

4 Yilun Hao, Yang Zhang, and Chuchu Fan. Planning anything with rigor: General-purpose zero-shot planning with llm-based formalized programming. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

5 HuggingFace. . `https://huggingface.co`, 2026. [Online; accessed 21-January-2026].

6 Alexey Ignatiev, António Morgado, and João Marques-Silva. PySAT: A python toolkit for prototyping with SAT oracles. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 428–437. Springer, 2018. `doi:10.1007/978-3-319-94144-8_26`.

7 Alexey Ignatiev, António Morgado, and João Marques-Silva. RC2: an efficient MaxSAT solver. *J. Satisf. Boolean Model. Comput.*, 11(1):53–64, 2019.

8 Chu Min Li and Felip Manya. Maxsat, hard and soft constraints. In *Handbook of satisfiability*, pages 613–631. IOS Press, 2009.

9 Meta. Llama-3.3 `https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_3/`, 2024. Released: 2024-12.

10 OpenAI. GPT5.5 `https://openai.com/index/introducing-gpt-5-5/`, 2026. Released: 2026-04-23.

11 Liangming Pan, Alon Albalak, Xinyi Wang, and William Wang. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3806–3824, 2023.

12 Weichun Shi, Minghao Liu, Wanting Zhang, Langchen Shi, Fuqi Jia, Feifei Ma, and Jian Zhang. Constraintllm: A neuro-symbolic framework for industrial-level constraint programming. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 16010–16030, 2025.

Table 1 Acceptance rate (%) of LLM-only and LLM+`PySAT` approaches.

Problem	Model	Direct	CoT	PoT	PySAT	PySAT+Plan
MIS	Gemini-3.1-Pro	0.0	4.0	11.0	36.0	**56.0**
	GPT-5.5	0.0	2.0	13.0	32.0	**51.0**
	Llama3.3	0.0	1.0	5.0	18.0	**24.0**
	Qwen3-Coder	0.0	0.0	6.0	15.0	**20.0**
Scheduling	Gemini-3.1-Pro	0.0	1.0	14.0	44.0	**59.0**
	GPT-5.5	0.0	0.0	9.0	41.0	**56.0**
	Llama3.3	0.0	0.0	3.0	**9.0**	8.0
	Qwen3-Coder	0.0	0.0	0.0	**22.0**	4.0
Set cover	Gemini-3.1-Pro	34.0	36.0	46.0	79.0	**87.0**
	GPT-5.5	29.0	33.0	45.0	76.0	**82.0**
	Llama3.3	14.0	37.0	41.0	**62.0**	45.0
	Qwen3-Coder	17.0	27.0	25.0	**49.0**	29.0

13 An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

14 Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. Satlm: Satisfiability-aided language models using declarative prompting. *Advances in Neural Information Processing Systems*, 36:45548–45580, 2023.

A Experimental Setup

All experiments were run using NVIDIA A100-SXM4 graphics card with 80GB of memory and 256GB of RAM, using a time limit of 5 minutes. For the MaxSAT oracle, RC2 [7] from the `PySAT` toolkit [6] was used, since `PySAT` is a Python API for several SAT/MaxSAT algorithms.

B Large Language Models (LLMs)

In our evaluation, we consider four LLMs: two open-source models and two closed-access models. For the open-source models, we select LLMs available on Hugging Face [5]: Alibaba's Qwen3-Coder [13] (30B) and Meta's Llama3.3 [9] (70B). These specific model sizes were chosen as they correspond to the most recent and largest available versions of each model that can be run on the GPU infrastructure used in our experiments. For the closed-access models, we evaluate OpenAI's GPT-5.5 [10] and Google's Gemini-3.1-Pro [2], as these are among the most widely adopted state-of-the-art proprietary models. Moreover, to ensure consistency across experiments, the temperature of all models was set to zero.

C Experimental Results

Table 1 reports acceptance rates for direct LLM solving, chain-of-thought (CoT), program-of-thought (PoT), and two `PySAT` based configurations. Bold entries indicate the best-performing configuration for each model and benchmark family. A solution is accepted only if it is both feasible and optimal with respect to an independently verified reference encoding.

D Implementation Notes

A practical implementation separates the system into four components. The first component stores the natural language task and the required output schema. The second component queries the LLM, either directly for code or first for an encoding plan. The third component executes the generated Python program in a sandbox with `PySAT` installed. The fourth component parses the output and checks that it is well formed.

When a reference encoding is available, validation can additionally check feasibility and objective value. For example, in a maximum independent set task, validation checks that no selected vertices are adjacent and that the reported set has optimal cardinality. In a set cover task, validation checks that all required elements are covered and that the reported cover has optimal cost. These checks are separate from the generated code and are used only for evaluation or debugging.