Solving MaxSAT Problems from Natural Language Descriptions with LLMs and PySAT

Pedro Orvalho [1*] Marta Kwiatkowska [2] Guillem Alenyà [3] Felip Manyà [1]

[1] Institut d'Investigació en Intelligència Artificial (IIIA-CSIC), Barcelona, Spain
[2] Department of Computer Science, University of Oxford, UK
[3] Institut de Robòtica i Informàtica Industrial (IRI-CSIC-UPC), Barcelona, Spain

The 2nd Workshop on LLMs meet Constraint Solving, **LLM-Solve @ FLoC 2026**
Lisbon, 19 July, 2026

*Work conducted while Pedro Orvalho was a visiting researcher at the University of Oxford, supported by the ELSA Mobility Program.

Users can describe problems, not encodings

Natural language input

"Choose a compatible set of projects. Some pairs cannot be selected together. Prefer higher value projects, but satisfy all hard restrictions."

- Descriptions mention objects, conflicts, capacities, coverage and preferences.

Users can describe problems, not encodings

Natural language input

"Choose a compatible set of projects. Some pairs cannot be selected together. Prefer higher value projects, but satisfy all hard restrictions."

- Descriptions mention objects, conflicts, capacities, coverage and preferences.
- Domain users often know the problem, but not the formal encoding.

Users can describe problems, not encodings

Natural language input

"Choose a compatible set of projects. Some pairs cannot be selected together. Prefer higher value projects, but satisfy all hard restrictions."

- Descriptions mention objects, conflicts, capacities, coverage and preferences.
- Domain users often know the problem, but not the formal encoding.

Formal MaxSAT model

Variables, hard clauses, weighted soft clauses, solver calls, model decoding, output schema, validation.

- Powerful and exact.

Users can describe problems, not encodings

Natural language input

"Choose a compatible set of projects. Some pairs cannot be selected together. Prefer higher value projects, but satisfy all hard restrictions."

- Descriptions mention objects, conflicts, capacities, coverage and preferences.
- Domain users often know the problem, but not the formal encoding.

Formal MaxSAT model

Variables, hard clauses, weighted soft clauses, solver calls, model decoding, output schema, validation.

- Powerful and exact.
- Difficult to write correctly by hand.

Users can describe problems, not encodings

Natural language input

"Choose a compatible set of projects. Some pairs cannot be selected together. Prefer higher value projects, but satisfy all hard restrictions."

- Descriptions mention objects, conflicts, capacities, coverage and preferences.
- Domain users often know the problem, but not the formal encoding.

Formal MaxSAT model

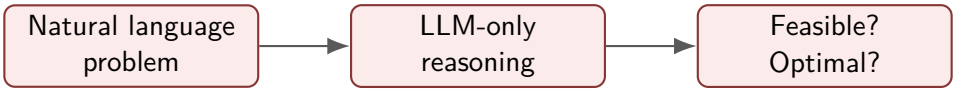
Variables, hard clauses, weighted soft clauses, solver calls, model decoding, output schema, validation.

- Powerful and exact.
- Difficult to write correctly by hand.

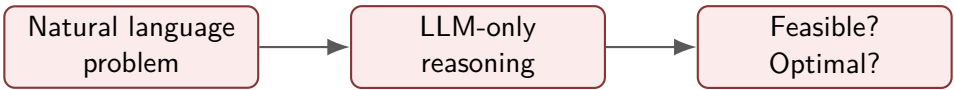
Main message

Use the LLM as a *modelling assistant*, not as the optimiser.

LLMs are fluent, but not reliable optimisers

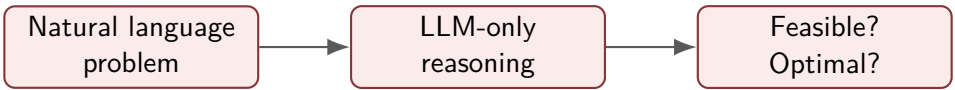


LLMs are fluent, but not reliable optimisers



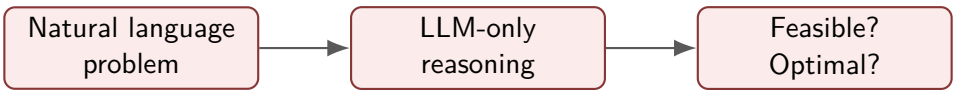
- A answer can violate hard constraints.

LLMs are fluent, but not reliable optimisers



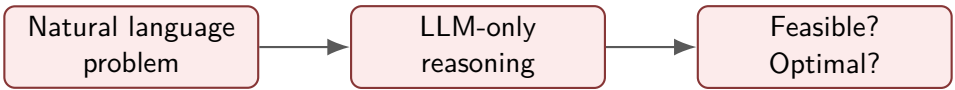
- A answer can violate hard constraints.
- A feasible answer can still be suboptimal.

LLMs are fluent, but not reliable optimisers



- A answer can violate hard constraints.
- A feasible answer can still be suboptimal.
- Chain-of-thought does not give an independent certificate of correctness.

LLMs are fluent, but not reliable optimisers

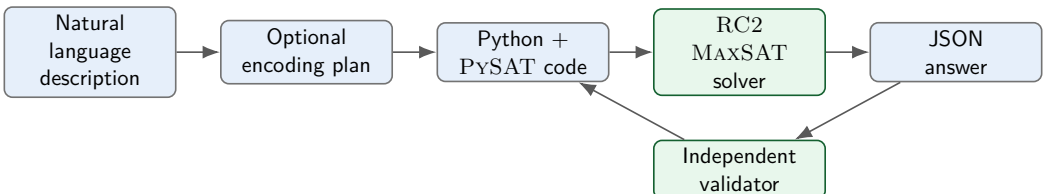


- A answer can violate hard constraints.
- A feasible answer can still be suboptimal.
- Chain-of-thought does not give an independent certificate of correctness.

Design choice

Separate **semantic parsing and modelling** from **exact optimisation**.

Our pipeline



- LLM translates the problem into a weighted partial MaxSAT instance.
- RC2 optimises the generated formula exactly.
- Validation checks feasibility and optimality against a canonical encoding.

What the LLM is asked to produce

Encoding plan

- decision variables

```python
from pysat.formula import WCNF
from pysat.examples.rc2 import RC2

wcnf = WCNF()
# hard: incompatible items cannot both be selected
wcnf.append([-x[i], -x[j]])
# soft: prefer selecting valuable items
wcnf.append([x[i]], weight=value[i])

with RC2(wcnf) as rc2:
    model = rc2.compute()
print(decode_as_json(model))
```

What the LLM is asked to produce

Encoding plan

- decision variables
- hard clauses

```python
from pysat.formula import WCNF
from pysat.examples.rc2 import RC2

wcnf = WCNF()
# hard: incompatible items cannot both be selected
wcnf.append([-x[i], -x[j]])
# soft: prefer selecting valuable items
wcnf.append([x[i]], weight=value[i])

with RC2(wcnf) as rc2:
    model = rc2.compute()
print(decode_as_json(model))
```

What the LLM is asked to produce

Encoding plan

- decision variables
- hard clauses
- weighted soft clauses

```python
from pysat.formula import WCNF
from pysat.examples.rc2 import RC2

wcnf = WCNF()
# hard: incompatible items cannot both be selected
wcnf.append([-x[i], -x[j]])
# soft: prefer selecting valuable items
wcnf.append([x[i]], weight=value[i])

with RC2(wcnf) as rc2:
    model = rc2.compute()
print(decode_as_json(model))
```

What the LLM is asked to produce

Encoding plan

- decision variables
- hard clauses
- weighted soft clauses
- output schema

```python
from pysat.formula import WCNF
from pysat.examples.rc2 import RC2

wcnf = WCNF()
# hard: incompatible items cannot both be selected
wcnf.append([-x[i], -x[j]])
# soft: prefer selecting valuable items
wcnf.append([x[i]], weight=value[i])

with RC2(wcnf) as rc2:
    model = rc2.compute()
print(decode_as_json(model))
```

What the LLM is asked to produce

Encoding plan

- decision variables
- hard clauses
- weighted soft clauses
- output schema

Generated program

Builds a WCNF formula, invokes RC2 using PySAT, decodes the model, and prints a prescribed JSON object.

```python
from pysat.formula import WCNF
from pysat.examples.rc2 import RC2

wcnf = WCNF()
# hard: incompatible items cannot both be selected
wcnf.append([-x[i], -x[j]])
# soft: prefer selecting valuable items
wcnf.append([x[i]], weight=value[i])

with RC2(wcnf) as rc2:
    model = rc2.compute()
print(decode_as_json(model))
```

Experimental setting

Benchmarks

- Maximum independent set

Evaluation

Compared methods

Experimental setting

Benchmarks

- Maximum independent set
- Scheduling

Evaluation

Compared methods

Experimental setting

Benchmarks

- Maximum independent set
- Scheduling
- Set cover

Evaluation

Compared methods

Experimental setting

Benchmarks

- Maximum independent set
- Scheduling
- Set cover

Evaluation

- 100 instances per family

Compared methods

Experimental setting

Benchmarks

- Maximum independent set
- Scheduling
- Set cover

Evaluation

- 100 instances per family
- accepted iff feasible and optimal

Compared methods

Experimental setting

Benchmarks

- Maximum independent set
- Scheduling
- Set cover

Evaluation

- 100 instances per family
- accepted iff feasible and optimal
- canonical MaxSAT encoding used as reference solution

Compared methods

Experimental setting

Benchmarks

- Maximum independent set
- Scheduling
- Set cover

Evaluation

- 100 instances per family
- accepted iff feasible and optimal
- canonical MaxSAT encoding used as reference solution

Compared methods

- Direct answer

Experimental setting

Benchmarks

- Maximum independent set
- Scheduling
- Set cover

Evaluation

- 100 instances per family
- accepted iff feasible and optimal
- canonical MaxSAT encoding used as reference solution

Compared methods

- Direct answer
- Chain-of-thought

Experimental setting

Benchmarks

- Maximum independent set
- Scheduling
- Set cover

Evaluation

- 100 instances per family
- accepted iff feasible and optimal
- canonical MaxSAT encoding used as reference solution

Compared methods

- Direct answer
- Chain-of-thought
- Program-of-thought

Experimental setting

Benchmarks

- Maximum independent set
- Scheduling
- Set cover

Evaluation

- 100 instances per family
- accepted iff feasible and optimal
- canonical MaxSAT encoding used as reference solution

Compared methods

- Direct answer
- Chain-of-thought
- Program-of-thought
- MaxSAT via PySAT

Experimental setting

Benchmarks

- Maximum independent set
- Scheduling
- Set cover

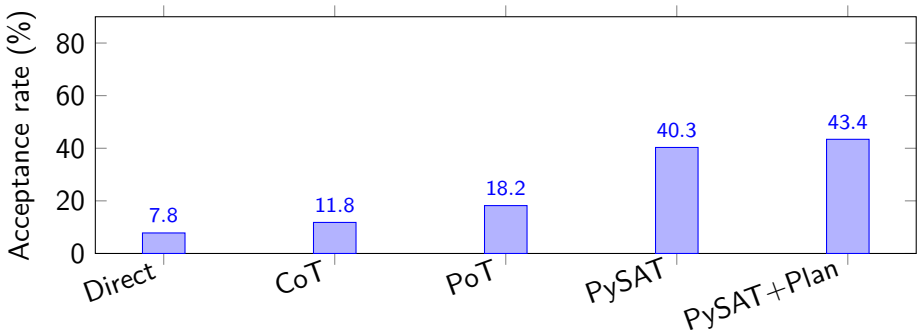
Evaluation

- 100 instances per family
- accepted iff feasible and optimal
- canonical MaxSAT encoding used as reference solution

Compared methods

- Direct answer
- Chain-of-thought
- Program-of-thought
- MaxSAT via PySAT
- MaxSAT via PySAT + plan

R1: Solver-backed modelling changes the game



Average over all problems and models reported in the workshop paper.

R2: Gains are strongest on hard families

Family	Model	Direct	CoT	PoT	PySAT	PySAT +Plan
Set cover	Gemini-3.1-Pro	34	36	46	79	**87**
Set cover	GPT-5.5	29	33	45	76	**82**
Scheduling	Gemini-3.1-Pro	0	1	14	44	**59**
Scheduling	GPT-5.5	0	0	9	41	**56**
MIS	Gemini-3.1-Pro	0	4	11	36	**56**
MIS	GPT-5.5	0	2	13	32	**51**

R2: Gains are strongest on hard families

Family	Model	Direct	CoT	PoT	PySAT	PySAT +Plan
Set cover	Gemini-3.1-Pro	34	36	46	79	**87**
Set cover	GPT-5.5	29	33	45	76	**82**
Scheduling	Gemini-3.1-Pro	0	1	14	44	**59**
Scheduling	GPT-5.5	0	0	9	41	**56**
MIS	Gemini-3.1-Pro	0	4	11	36	**56**
MIS	GPT-5.5	0	2	13	32	**51**

Takeaway

The key improvement is not more reasoning text; it is executable, solver-verifiable MaxSAT modelling using PySAT.

Why planning can help - and hurt

Helpful when...

- variables are named explicitly;

Harmful when...

Why planning can help - and hurt

Helpful when...

- variables are named explicitly;
- hard and soft constraints are separated;

Harmful when...

Why planning can help - and hurt

Helpful when...

- variables are named explicitly;
- hard and soft constraints are separated;
- the output schema is fixed before coding;

Harmful when...

Why planning can help - and hurt

Helpful when...

- variables are named explicitly;
- hard and soft constraints are separated;
- the output schema is fixed before coding;
- the model can faithfully implement its plan.

Harmful when...

Why planning can help - and hurt

Helpful when...

- variables are named explicitly;
- hard and soft constraints are separated;
- the output schema is fixed before coding;
- the model can faithfully implement its plan.

Harmful when...

- the plan contains a semantic mistake;

Why planning can help - and hurt

Helpful when...

- variables are named explicitly;
- hard and soft constraints are separated;
- the output schema is fixed before coding;
- the model can faithfully implement its plan.

Harmful when...

- the plan contains a semantic mistake;
- code generation propagates that mistake;

Why planning can help - and hurt

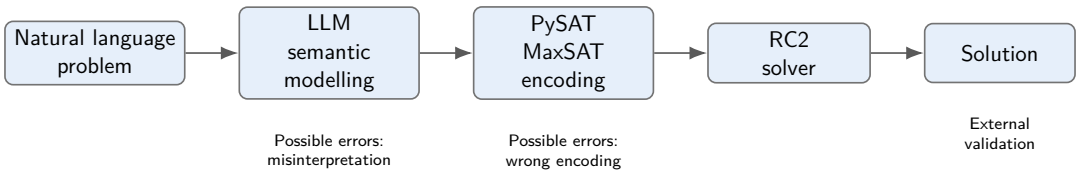
Helpful when...

- variables are named explicitly;
- hard and soft constraints are separated;
- the output schema is fixed before coding;
- the model can faithfully implement its plan.

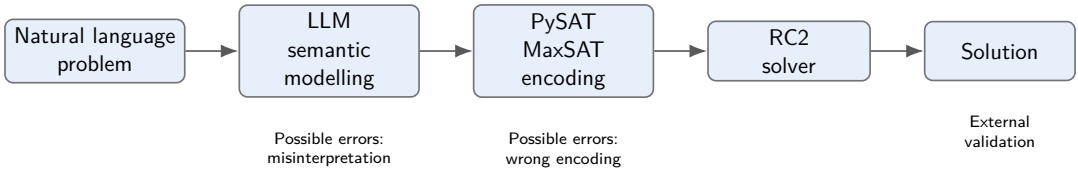
Harmful when...

- the plan contains a semantic mistake;
- code generation propagates that mistake;
- repair loops mask, but do not fix, incorrect modelling.

Why a neuro-symbolic approach?



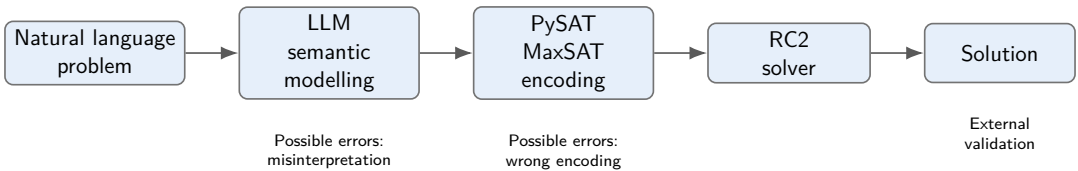
Why a neuro-symbolic approach?



Key takeaway

- The solver guarantees an *optimal solution* for the generated MaxSAT model.

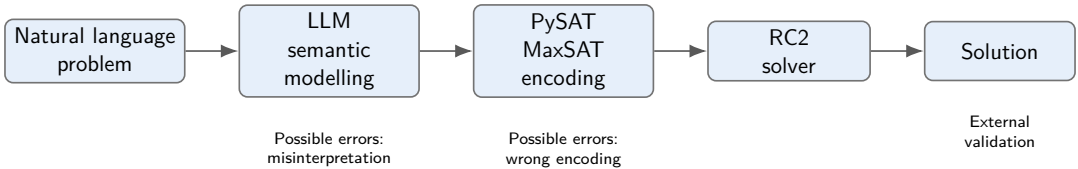
Why a neuro-symbolic approach?



Key takeaway

- The solver guarantees an *optimal solution* for the generated MaxSAT model.
- Validation checks whether the generated model matches the intended semantics.

Why a neuro-symbolic approach?



Key takeaway

- The solver guarantees an *optimal solution* for the generated MaxSAT model.
- Validation checks whether the generated model matches the intended semantics.
- Separating modelling from solving makes this process easier to debug than end-to-end LLM reasoning.

Limitations

- Controlled natural language is easier than real user input.

Limitations

- Controlled natural language is easier than real user input.
- The LLM still has to model the intended problem correctly.

Limitations

- Controlled natural language is easier than real user input.
- The LLM still has to model the intended problem correctly.
- Planning and repair can introduce error propagation.

Limitations

- Controlled natural language is easier than real user input.
- The LLM still has to model the intended problem correctly.
- Planning and repair can introduce error propagation.
- Current evaluation focuses on three benchmark families.

Open questions for discussion

1. What validation should be required before trusting a natural-language encoding?

Open questions for discussion

1. What validation should be required before trusting a natural-language encoding?
2. How can we verify encoding plans generated by LLMs?

Open questions for discussion

1. What validation should be required before trusting a natural-language encoding?
2. How can we verify encoding plans generated by LLMs?
3. How can we verify intermediate MaxSAT encodings (i.e., variables, constraints)?

Open questions for discussion

1. What validation should be required before trusting a natural-language encoding?
2. How can we verify encoding plans generated by LLMs?
3. How can we verify intermediate MaxSAT encodings (i.e., variables, constraints)?
4. How should benchmarks capture ambiguity, evolving preferences, and inconsistent requirements?

Conclusion

- Natural language can make MaxSAT accessible to non-experts.

Conclusion

- Natural language can make MaxSAT accessible to non-experts.
- LLM-only optimisation remains unreliable for constrained tasks.

Conclusion

- Natural language can make MaxSAT accessible to non-experts.
- LLM-only optimisation remains unreliable for constrained tasks.
- LLMs are useful as front-ends that generate PySAT encodings.

Conclusion

- Natural language can make MaxSAT accessible to non-experts.
- LLM-only optimisation remains unreliable for constrained tasks.
- LLMs are useful as front-ends that generate PySAT encodings.
- RC2 provides exact optimisation; independent validation checks semantic correctness.

Conclusion

- Natural language can make MaxSAT accessible to non-experts.
- LLM-only optimisation remains unreliable for constrained tasks.
- LLMs are useful as front-ends that generate PySAT encodings.
- RC2 provides exact optimisation; independent validation checks semantic correctness.

Conclusion

- Natural language can make MaxSAT accessible to non-experts.
- LLM-only optimisation remains unreliable for constrained tasks.
- LLMs are useful as front-ends that generate PySAT encodings.
- RC2 provides exact optimisation; independent validation checks semantic correctness.

Use LLMs to write MaxSAT models using PySAT.

Let solvers solve the model.

Thank you

Questions?

This work was conducted during a University of Oxford visit supported by the ELSA Mobility Program. This work was supported by grant PID2022-139835NB-C21 funded by MCIN/AEI/10.13039/501100011033 and by ERDF, EU; and 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).

Full acceptance table from extended version

Family	Model	Direct	CoT	PoT	No plan	With plan
MIS	Gemini-3.1-Pro	0	4	11	36	56
MIS	GPT-5.5	0	2	13	32	51
MIS	Llama3.3	0	1	5	18	24
MIS	Qwen3-Coder	0	0	6	15	20
Scheduling	Gemini-3.1-Pro	0	1	14	44	59
Scheduling	GPT-5.5	0	0	9	41	56
Scheduling	Llama3.3	0	0	3	9	8
Scheduling	Qwen3-Coder	0	0	0	22	4
Set cover	Gemini-3.1-Pro	34	36	46	79	87
Set cover	GPT-5.5	29	33	45	76	82
Set cover	Llama3.3	14	37	41	62	45
Set cover	Qwen3-Coder	17	27	25	49	29

Weighted partial MaxSAT reminder

Given Boolean variables $x_1, \ldots, x_n$:

- **Hard clauses** must be satisfied.
- **Soft clauses** have weights and may be violated at a cost.
- The goal is to satisfy all hard clauses and optimise the total satisfied soft weight.

Why it fits preference reasoning

Hard requirements and user preferences are represented in the same formalism, but with different semantics.