A First Prototype Involving Turbo for Solving LLM-Driven Generated Constraints?

NEURESA Workshop 2026

Clément Poncelet

University of Luxembourg

07/09/26

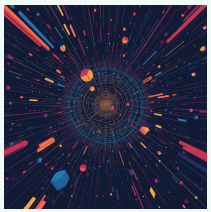
NEURESA

Embedded System Modeling

Cognifyer

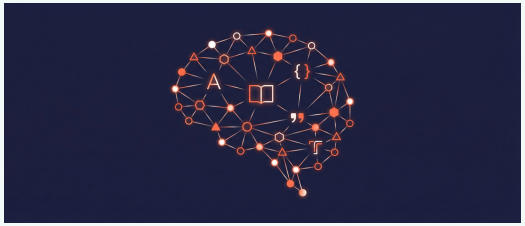
- Architecture design
- Industrial automation

Turbo



- Constraint solver
- GPU acceleration

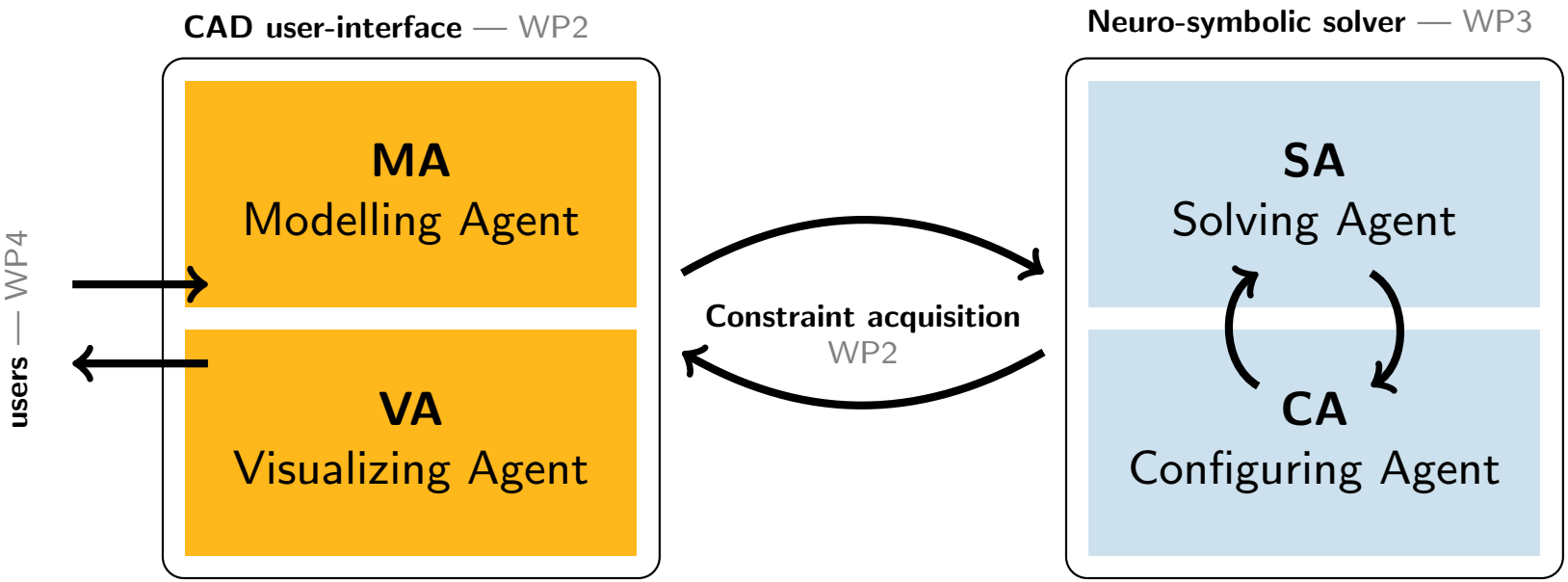
Large Language Models



- Text generation
- User assistance

Overview

Toward a LLM-guided assistant for neuro-symbolic solver on GPU!

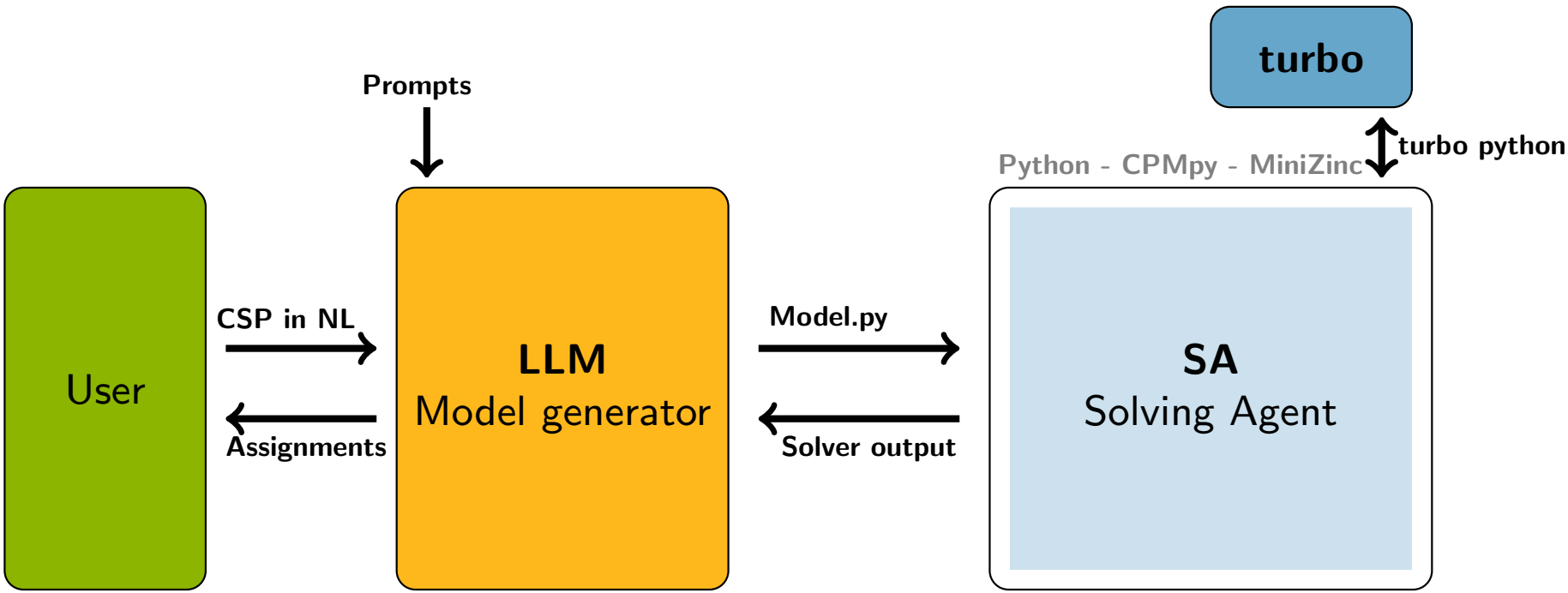


Objectives

- Democratize constraint network modeling.
- Incremental solving and user interaction.
- Visual representation of assignments.
- Automatic and on-the-fly configuration of solvers.

Step 1: Prototype

Generate a constraint network via LLM.

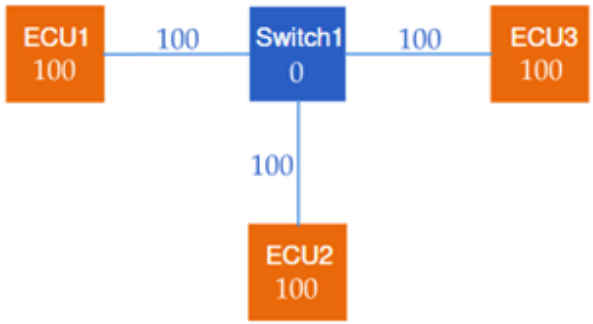


Implemented

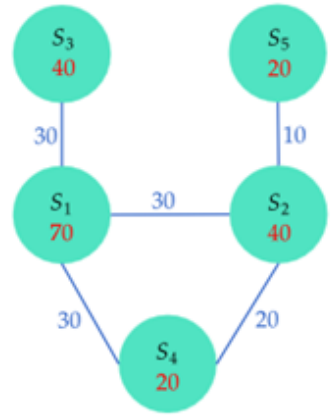
- Python extension for turbo (pybind11)
- CPMpy backend implentation with turbo (using MiniZinc)
- Connection to free LLM via OpenRouter

Demo

Demo

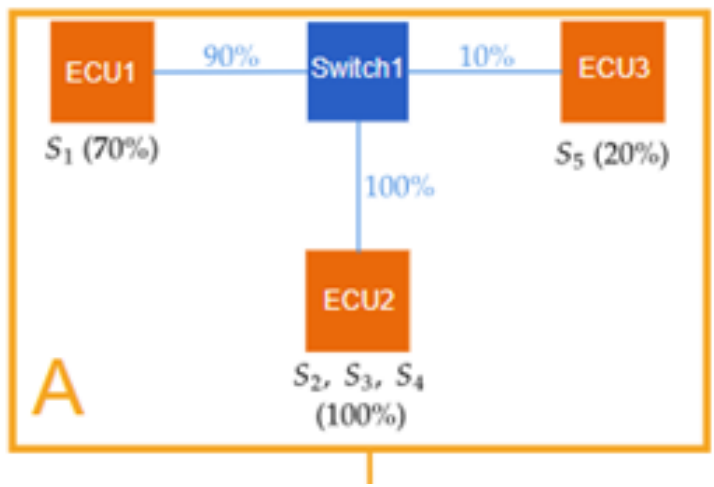


(a) Hardware graph with 3 ECUs and 1 switch. The values of *hc* are displayed in white and those of *lc* in blue.



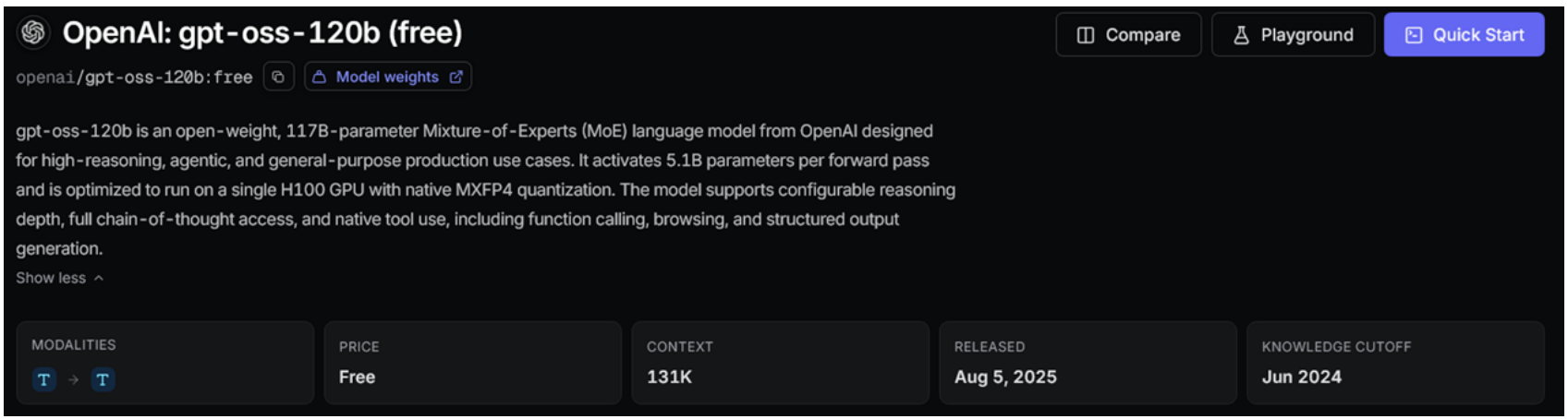
(b) Software graph with 5 services and 5 communications. The values of *sc* are displayed in red and those of *cc* in blue.

Figure 1 An example of the software deployment problem.



To complete

- Clean code and pass CPmPy tests
- User interaction for refining
  (with or without external functions)
- Limit?



Next Steps

Model Generation

- Sampling & self-verification
- Semantic correctness
- Solver configuration

Interpretability

- Reinforcement learning
- CSP patterns
- Size vs. effectiveness

Output Generation

- Visual representation of:
  - Assignments
  - State spaces

Constraint Acquisition

- Incremental solving

Model Generation

- Sampling & self-verification
- Semantic correctness
- Solver configuration

Model Generation

From DCP-Bench-Open.

Highlight

Evaluate LLM-driven constraint modelling.

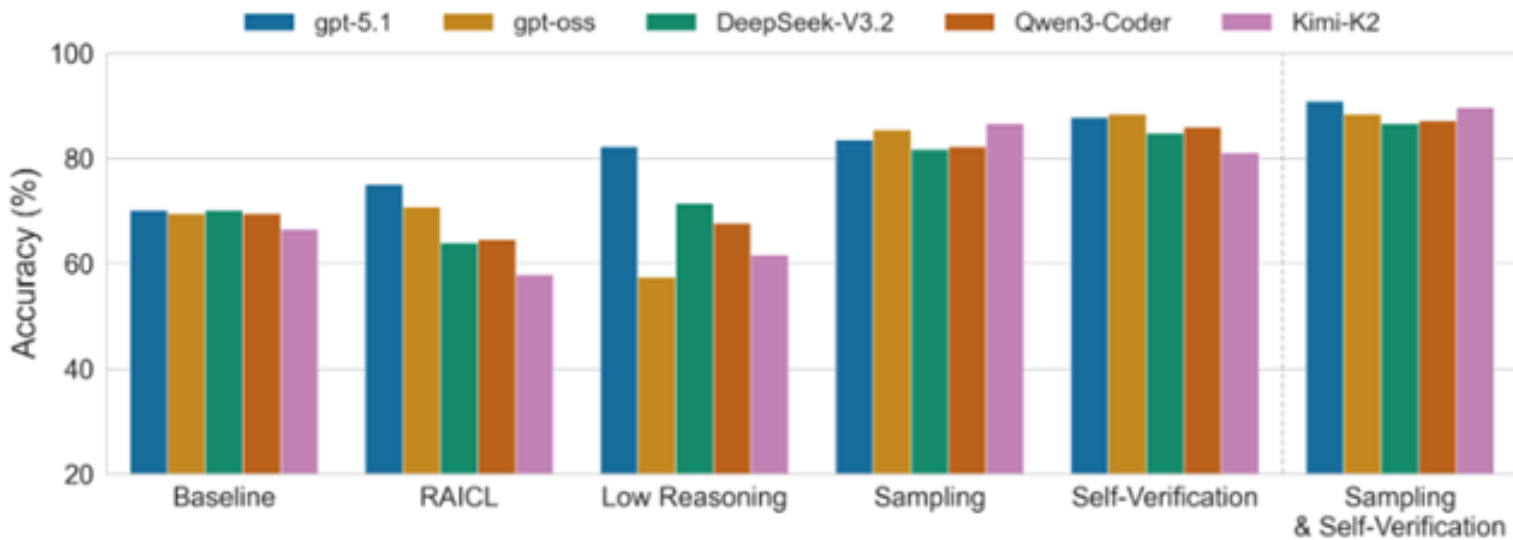


Fig. 7. Performance (Single Instance Accuracy) across different inference-time compute methods for each LLM using CPMpy.

Reference

K. Michailidis et al., *DCP-Bench-Open: Evaluating LLMs for Constraint Modelling of Discrete Combinatorial Problems*, arXiv 2026.

Model Generation

Sampling & Self-Verification

Highlight

Adding a loop for correcting model errors.

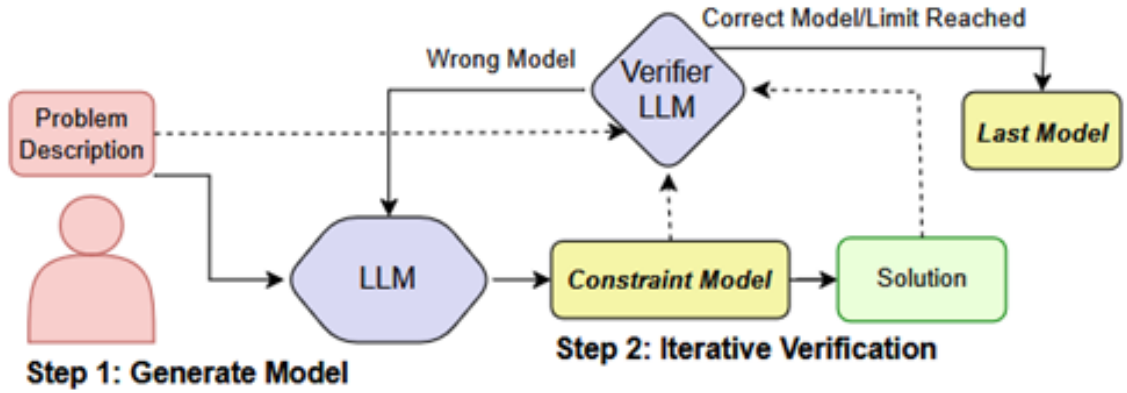


Fig. 3. Iterative Self-Verification: the generated CP model is verified iteratively; both the original problem description and produced solution are also provided back to the LLM. In this work, we use a single LLM for both model generation and verification (thus, self-verification).

Modeling errors : yields incorrect solutions or unsatisfiable models.

Model Generation

Semantic correctness.

Highlight

- Correcting syntactic errors handled by LLM
- Modeling errors ?

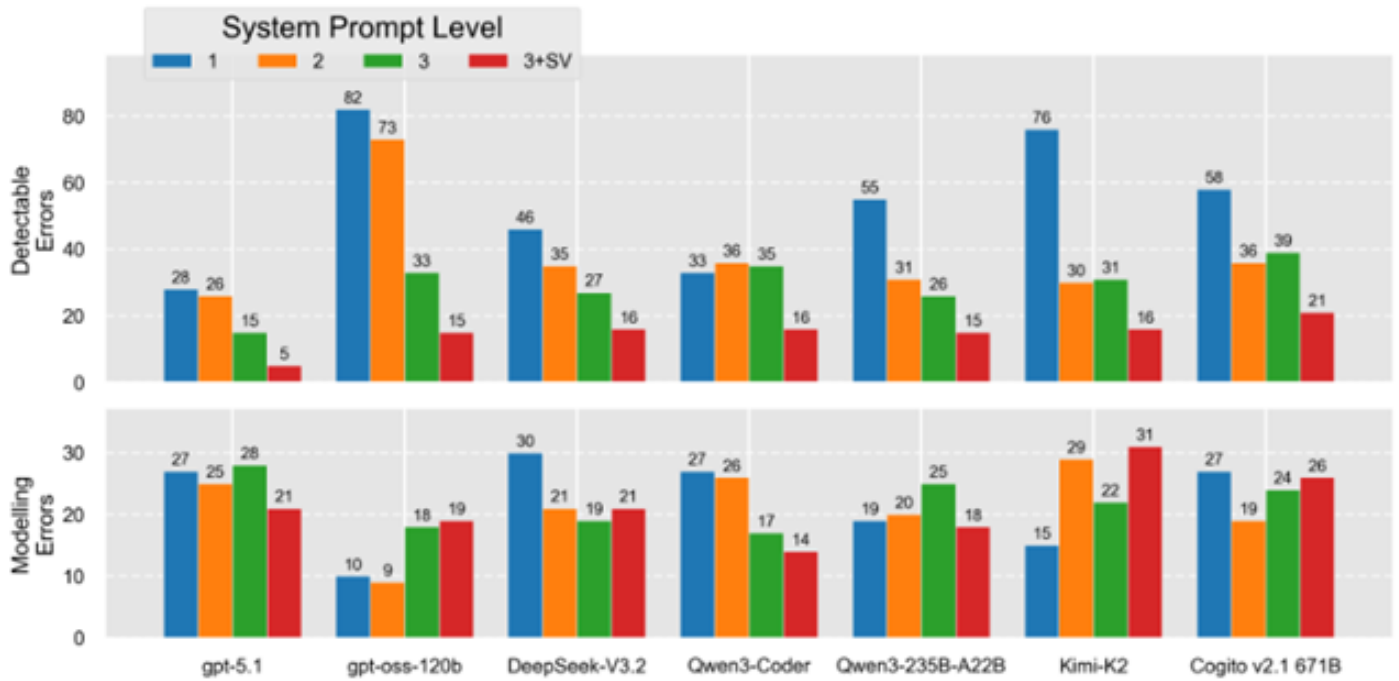


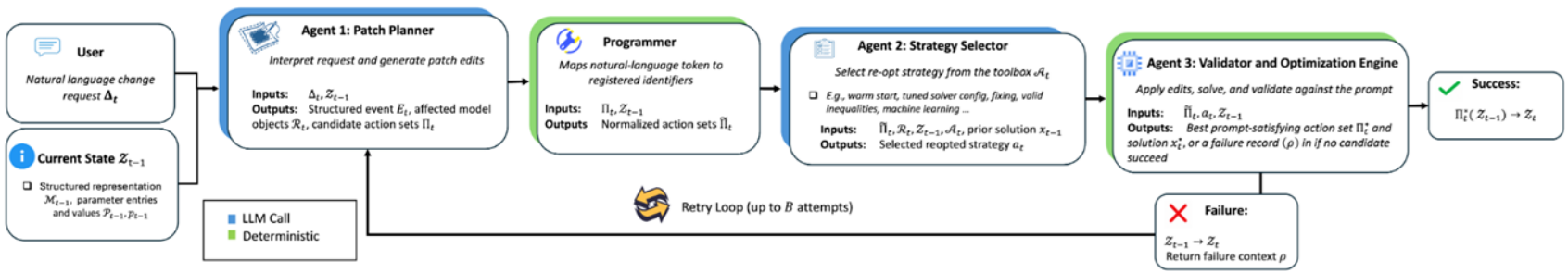
Fig. 6. Detectable and modelling errors with CPMpy. 3+SV stands for Self-Verification of a single repetition along with a system prompt level 3 (documentation). Importantly, the (green) bars for level 3 represent the same experimental runs as 3+SV (red), but report the errors prior to the self-verification iteration, for reproducibility.

Model Generation

Solver configuration.

Highlight

- Optimizing solving time.
- DSL for re-configuring the constraint network.



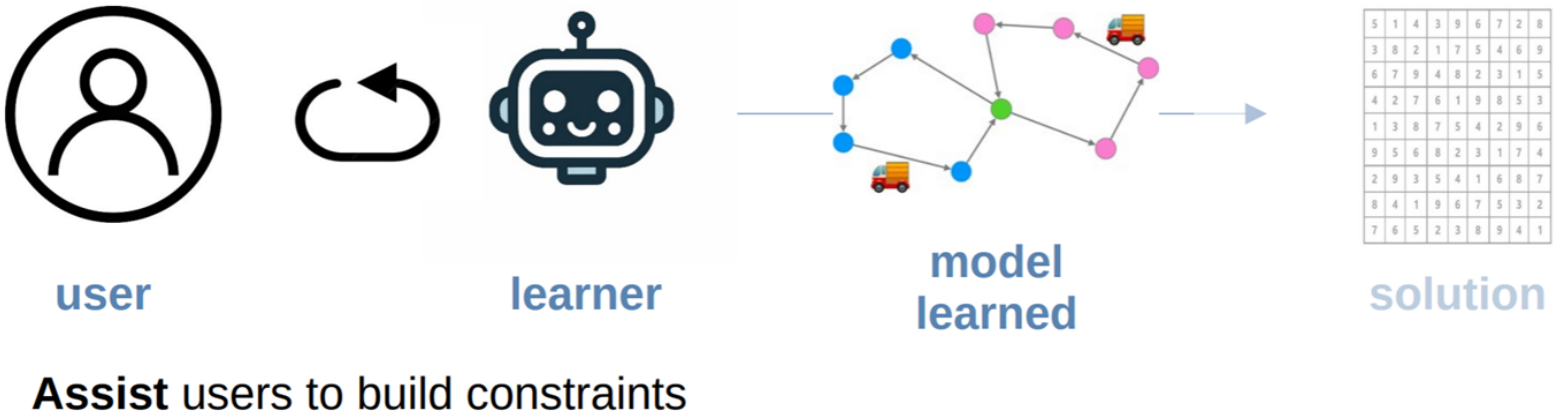
Reference

T. Ye et al., *Democratizing Large-Scale Re-Optimization with LLM-Guided Model Patches*, arXiv 2026.

Constraint Acquisition

- Incremental solving

Constraint Acquisition



The LLM can be used to:

- interpret user feedback from a model or (partial) assignments.
- adapt the model from external cost functions.

Reference

Y. Mechqrane et al., *Active Constraint Acquisition Using Large Language Models*, JAIR 2026.

Incremental solving

```
let mut solver = Solver::new();

solver.add_clause(&[!x, y]);
solver.add_clause(&[!y, z]);
assert_eq!(solver.solve().unwrap(), true);

solver.add_clause(&[!z, x]);
assert_eq!(solver.solve().unwrap(), true);
```

- Assumptions (temporary constraints).
- States.
- For Turbo: atomic API.

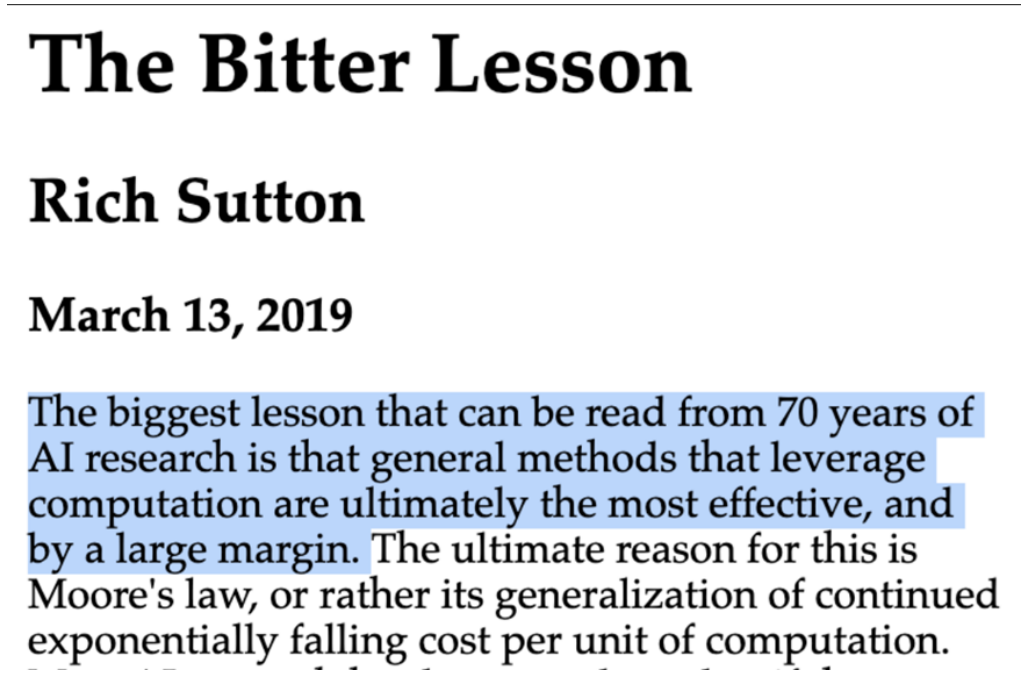
Output Generation

- Visual representation of:
  - Assignments
  - State spaces

Visual representation

Quacq - Web IHM

World Models



The Bitter Lesson

Rich Sutton

March 13, 2019

The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin. The ultimate reason for this is Moore's law, or rather its generalization of continued exponentially falling cost per unit of computation.

- Václav Volhejn
- Kyutai
- Video-zero-shot
- Le World Model

Interpretability

- Reinforcement learning
- CSP patterns
- Size vs. effectiveness

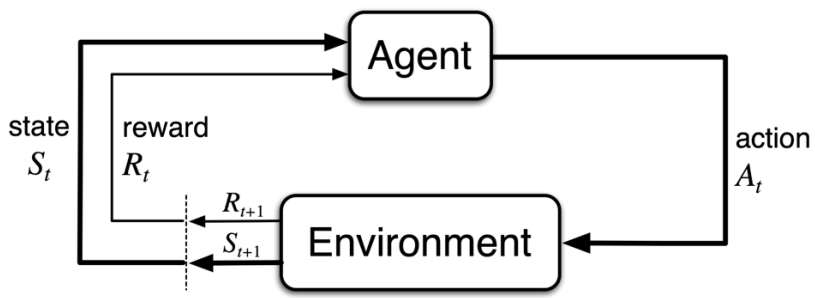
Interpretability

Eiffel-tower-llama

Sparse autoencoders:

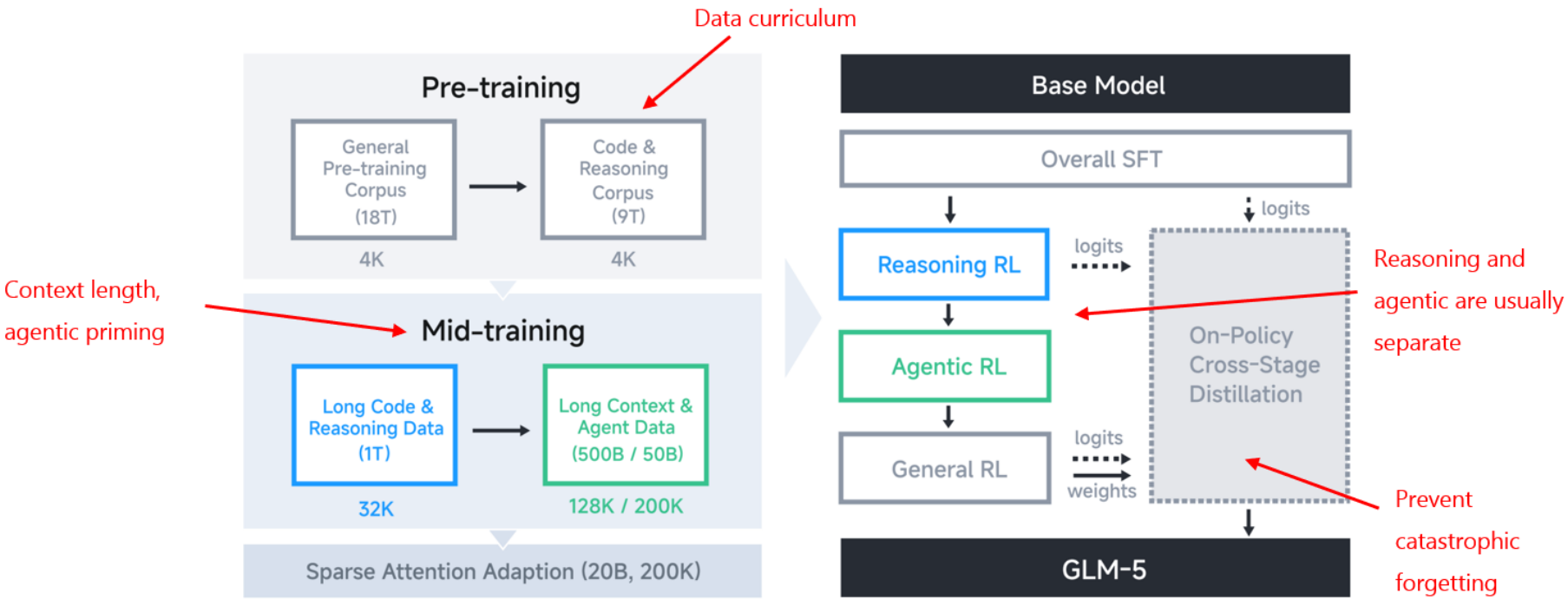
- Steer the model's behavior
- by modifying its activations at **inference**

What is an agent ?



Interpretability

Modern training pipelines



GLM-5 Tech report (Feb 2026): https://arxiv.org/pdf/2602.15763

Reinforcement Learning

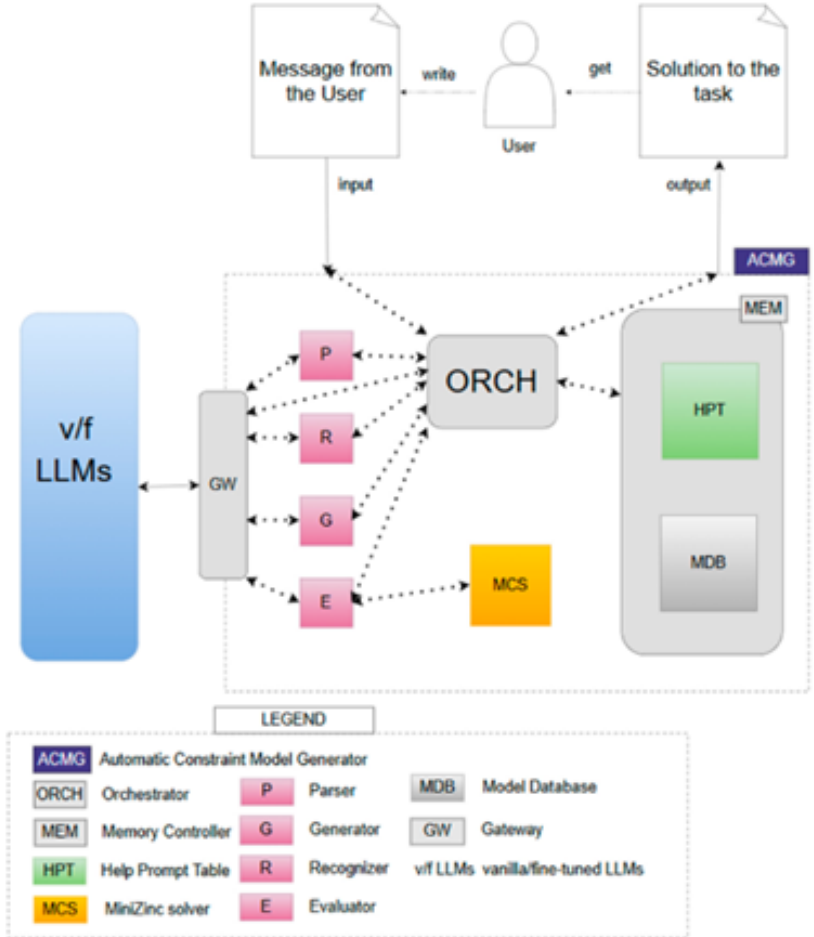


Figure 10. Architecture overview of the Automatic Constraint Model Generator (ACMG).

Reference

R. Penco et al., *Large Language Model-Driven Framework for Automated Constraint Model Generation in Configuration Problems*, Applied Sciences, 15(12), 6518, 2025.

Reinforcement Learning

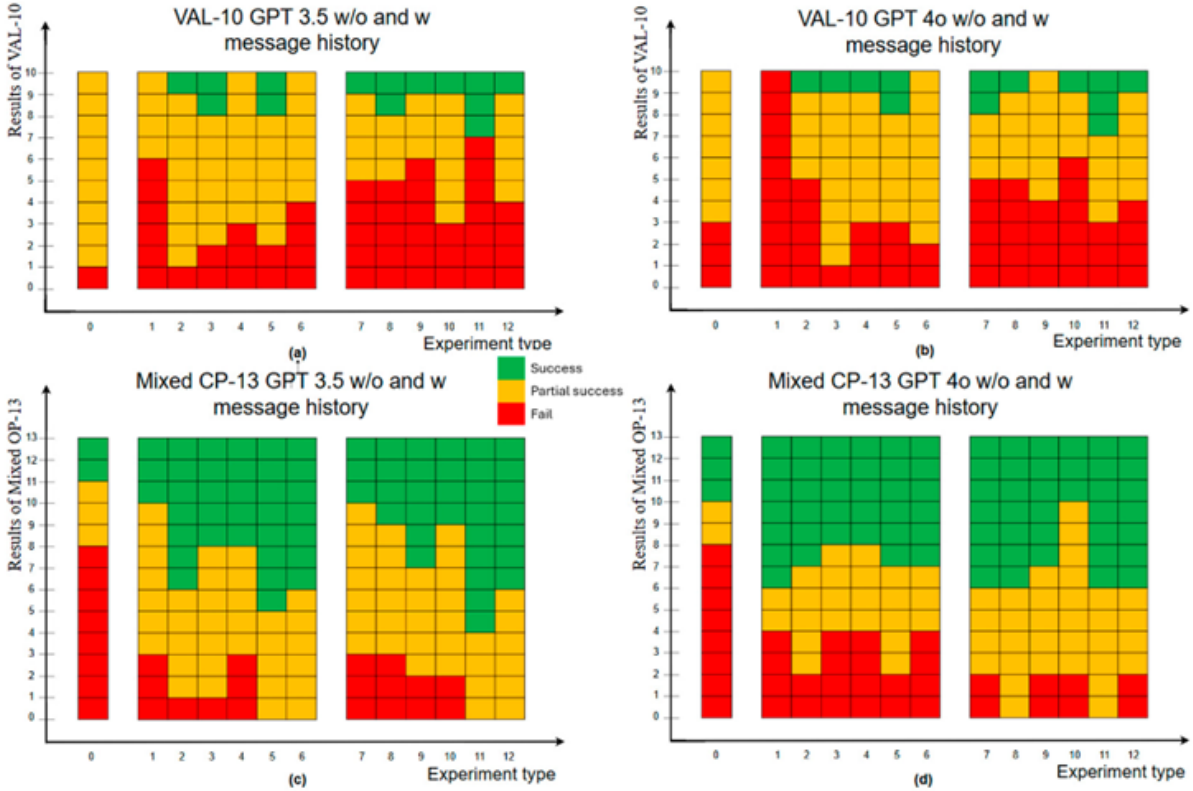


Figure 12. The results of the four different groups of experiments: **(a)** VAL-12 GPT 3.5 turbo; **(b)** VAL-12 GPT 4o; **(c)** Mixed-CP GPT 3.5 turbo; and **(d)** Mixed-CP GPT 4o.

Table A3. Ablation Configurations (Component Dataset Fine-Tuning View).

	V1 [1]	V2	V3	V4	V5	V6
vLLM [5]	P, R, G, E [3]	E	P, E	P, E	E	E
fLLM(CSPNER-50) [2]			R	R	R	R
fLLM(CSPNERMZC-50)		P, R, G		G		G
fLLM(MZC-50)			G		G	
fLLM(CSP-50)					P	P
fLLM(CSPNER-50)						
LLM Qty [4]	1	2	3	3	4	4

[1] VX—variant number for architectural configuration of ACMG; V1 is Variant 1. [2] fLLM(Dataset) represents the distinct LLM fine-tuned with our dataset. [3] P,R,G,E—Parser, Recognizer, Generator, and Evaluator, respectively. [4] LLM Qty—the distinct number of LLMs in that configuration variant. [5] Vanilla LLM.