LLM-Con: Large-Language Models for Constraint Modelling

Tutorial, IJCAI 2026, Bremen, Germany

Tias Guns

Professor
Dept. of Computer Science, KU Leuven
https://people.cs.kuleuven.be/~tias.guns/

Serdar Kadıoğlu

Adj. Assoc. Professor
Dept. of Computer Science, Brown
Group VP, AI Center of Excellence, Fidelity
skadio.github.io

Dimos Tsouros

Assistant Professor
Univ. of Western Macedonia
https://dimostsouros.github.io/

Tutorial Agenda

Session A: Introduction, data, evaluation (9:00 - 10:30)

A1) Introduction (30 min): Tias Guns

A2) Benchmarking & Evaluation (60 min): Dimos Tsouros

Break (10:30 - 11:00)

Session B: Modelling copilots (11:00 - 12:30)

B1) Modelling copilots (60 min): Serdar Kadioglu

B2) Wrap-up (30 min): Tias Guns

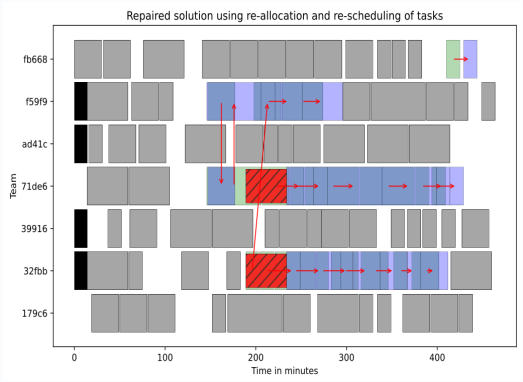
Session A.1

Introduction

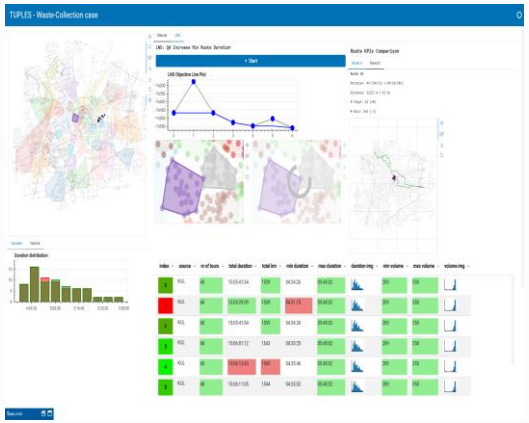
Presenter: Tias Guns

Combinatorial optimization

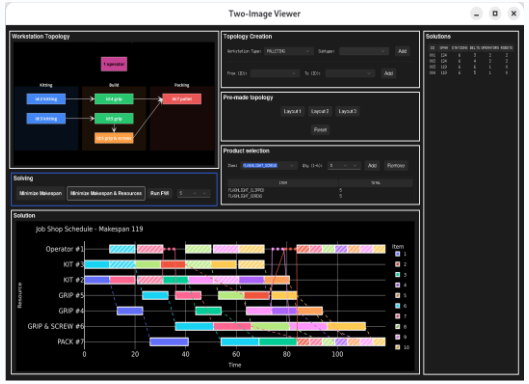
“Solving constrained satisfaction and optimization problems”



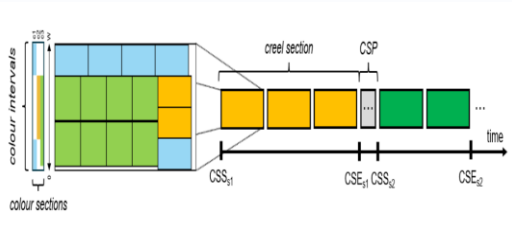
Workforce allocation



Waste collection routing



Assembly scheduling



Industrial weave production planning

Central role in AI and OR

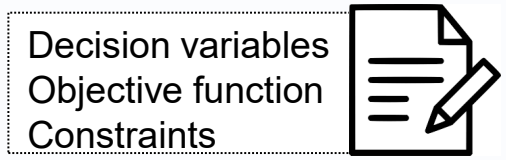
“Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it.”

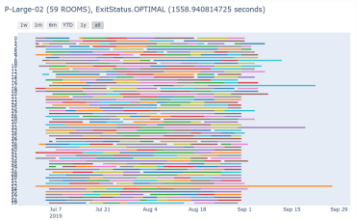
— E. Freuder, Constraints, 1997

Classic Model + Solve paradigm:

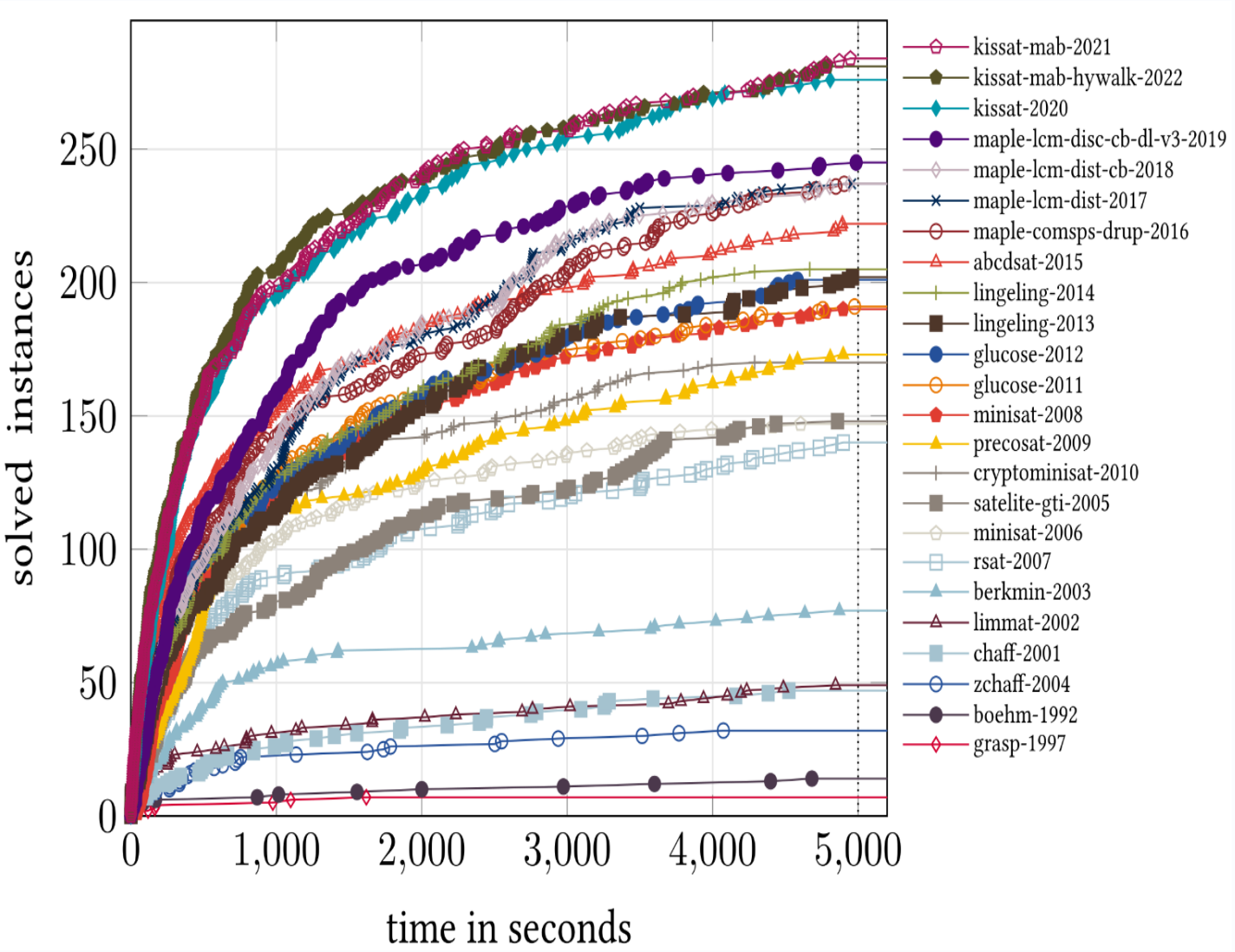
Model + Solve

Decision variables
Objective function
Constraints





Solve: Tremendous progress



MIP and SAT, 30+ years

MIP: ~10^6 algorithmic speedup since the early 90s
(Bixby; Koch et al., 2022).

SAT: cactus plot, 1992–2022 (left)
[Biere et al., SAT Museum 2023].

For many problems we care about, solve time is no longer a showstopper

+ many have good anytime behaviour

+ parallel portfolio solvers (gurobi, or-tools cp-sat, hexaly, ...)

Model: Combinatorial Problems

CSPs: Constraint Satisfaction Problem

A Constraint Satisfaction Problem is defined as a triple:

$$CSP = (X, D, C)$$

- **X**: Finite set of decision variables
- **D**: Domains assigning each variable admissible values
- **C**: Constraints mapping assignments to truth values

A **feasible solution** assigns all variables such that all constraints evaluate to *true*.

COPs: Constrained Optimization Problems

Optimization augment CSPs with an objective function:

$$COP = (X, D, C, O)$$

- **O:** Objective assigns cost/value to assignments.

An **optimal solution** minimizes or maximizes O while respecting all constraints.

Model: textbook example: Abbot's puzzle

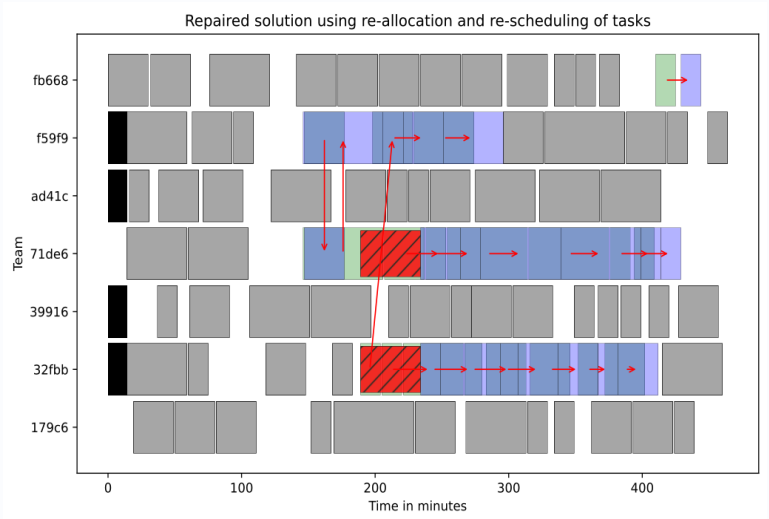
If 100 bushels of corn were distributed among 100 people such that each man received three bushels, each woman two, and each child half a bushel, and there are five times as many women as men, find the number of men, women, and children.

hakank / Alcuin.

dcp-bench.github.io/DCP-Bench-Open/problems/abbots_puzzle.html

```python
from cpmpy import *
import json

# Decision variables
men = intvar(0, 100, name="men")
women = intvar(0, 100, name="women")
children = intvar(0, 100, name="children")

# Model
model = Model([
    men + women + children == 100,   # 100 people
    men * 6 + women * 4 + children == 200,  # 100 bushels (x2)
    men * 5 == women                 # 5x as many women
])

model.solve()
print(json.dumps({
    "men": men.value(),
    "women": women.value(),
    "children": children.value()
}))
# {"men": 5, "women": 25, "children": 70}
```

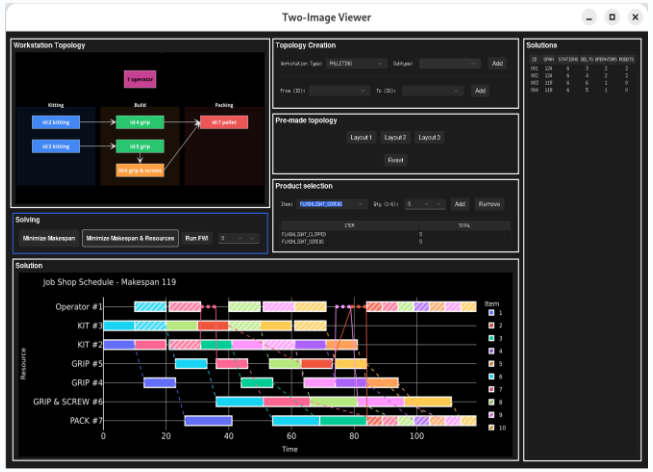
Real world examples (bigger models, same solvers)



Workforce Re-Allocation:

- Each task must start after its release time and finish by its deadline
- A team can only perform one *compatible* task at a time
- Tasks that belong together (e.g. moving the same part) must be done by the same team
- Some tasks must finish before others can start

Minimize nr of jobs moved, maximize spread across teams



Assembly scheduling:

- Each item requires a sequence of operations, to be done in order
- Each operation must be assigned to a workstation that can perform it
- A resource (workstation, employee, robot) can execute at most one operation at a time
- Items must be moved between workstations (by employee, robot or conveyor)

Minimize makespan and nr of resources used.

Model + Solve: Modelling requires expertise

1 **Domain knowledge**

What is actually allowed in this organisation?

2 **Solver formalisms**

CP vs MIP vs SMT vs (Max)SAT vs ...

3 **Mathematical skills**

Decision variables & viewpoints
Declarative constraints (arithmetic, logic, global constraints)
Efficiency (implied constraints, symmetry breaking, ...)

Model + Solve, in practice

1 Problem

In someone's head / in an organisation

2 Problem description

Natural language.
Already an interpretation

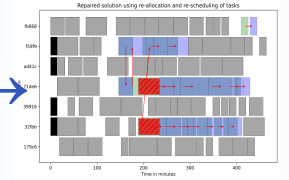
3 Model

```python
from cpmpy import *
import json

# Decision variables
men = intvar(0, 100, name="men")
women = intvar(0, 100, name="women")
children = intvar(0, 100, name="children")

# Model
model = Model([
    men + women + children == 100,   # 100 people
    men * 6 + women * 4 + children == 200,  # 100 bushels (x2)
    men * 5 == women                 # 5x as many women
])

model.solve()
print(json.dumps({
    "men": men.value(),
    "women": women.value(),
    "children": children.value()
}))
# {"men": 5, "women": 25, "children": 70}
```

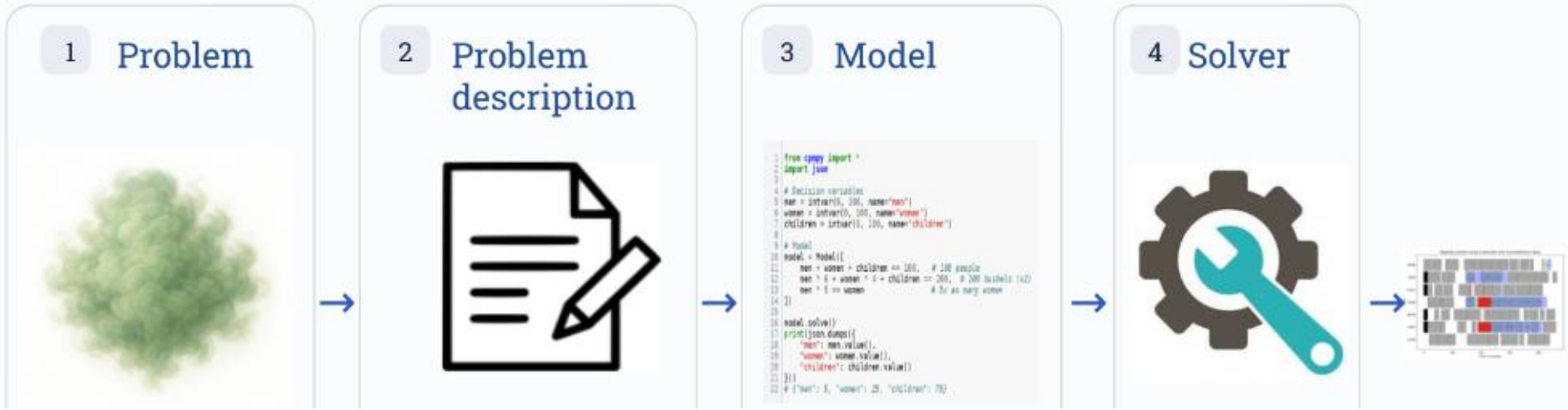
Formal language/ system API.
An encoding

4 Solver

Or-Tools CP-SAT, Gurobi, Hexaly, ...



Using LLMs to help solve CO problems?



Just solve it	‘Reasoning’ LLMs	**Does not scale**
Write the model	From a natural-language description	**This tutorial**
Tune the solver / Invent Algorithms	Parameters, from documentation / papers FunSearch, EoH, ReEvo, AlphaEvolve, …	**T15 on Sat***

T15 on Sat*: a complementary tutorial last Saturday: LLMs for Optimisation

Types of LLMs

Plain LLM:

- Pre-trained text generator
- Takes a ‘prompt’ as input, **completes** it by generating text one ‘token’ at a time.

Coding LLM:

- Pre-trained text generator, fine-tuned with lots of code examples and tasks

Agentic LLM:

- Pre-trained text generator that can call external tools
- (ex: internet search or execute programs, including self-written code)

Splashy LLMs for problem solving

1

AlphaEvolve, DeepMind, 2025

An evolutionary coding agent powered by large language models,
for (sub)algorithm discovery and algorithm optimization.

2

Claude Code/Cursor/DeepSeek coder/ …

Competitive programming; SWE-bench, …
Trained in agent coding hardness with Reinforcement Learning on problems with test-suites.

3

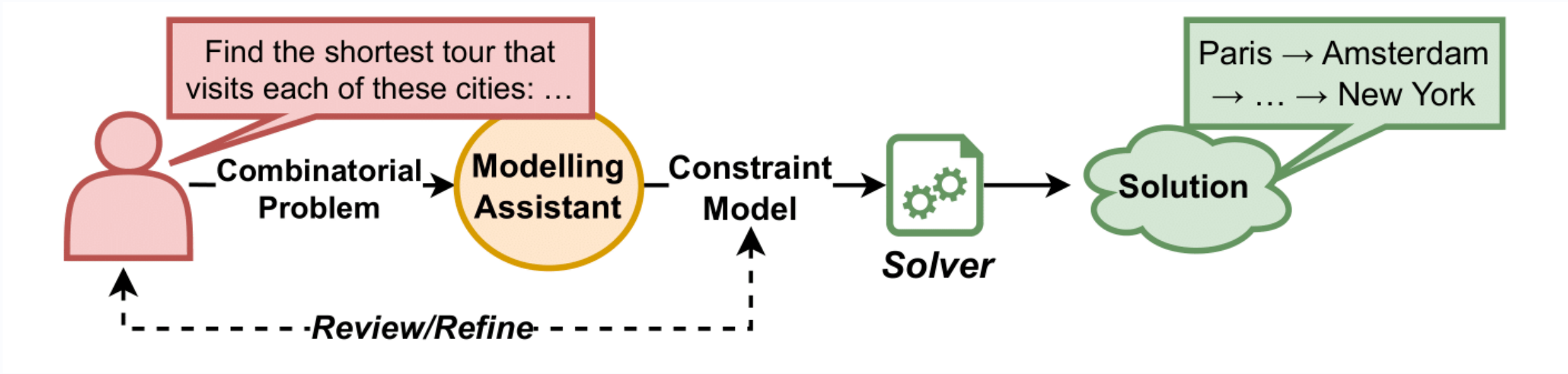
10 Proofs by OpenAI, 1 august 2026

10 mathematical proofs on long-standing open mathematics problems, with formally verified Lean proofs.

LLM assistants

- Chatbots and chat-based apps everywhere!
- Coding assistants (e.g. Claude Code, Cursor) excel in general coding tasks

What about modeling assistants?



Modelling CO = programming?

Imperative Programming

Python, C++, Java, ...

‘How’ things should be executed

Bugs tend to crash or fail a test

Lots of Python code on the web.

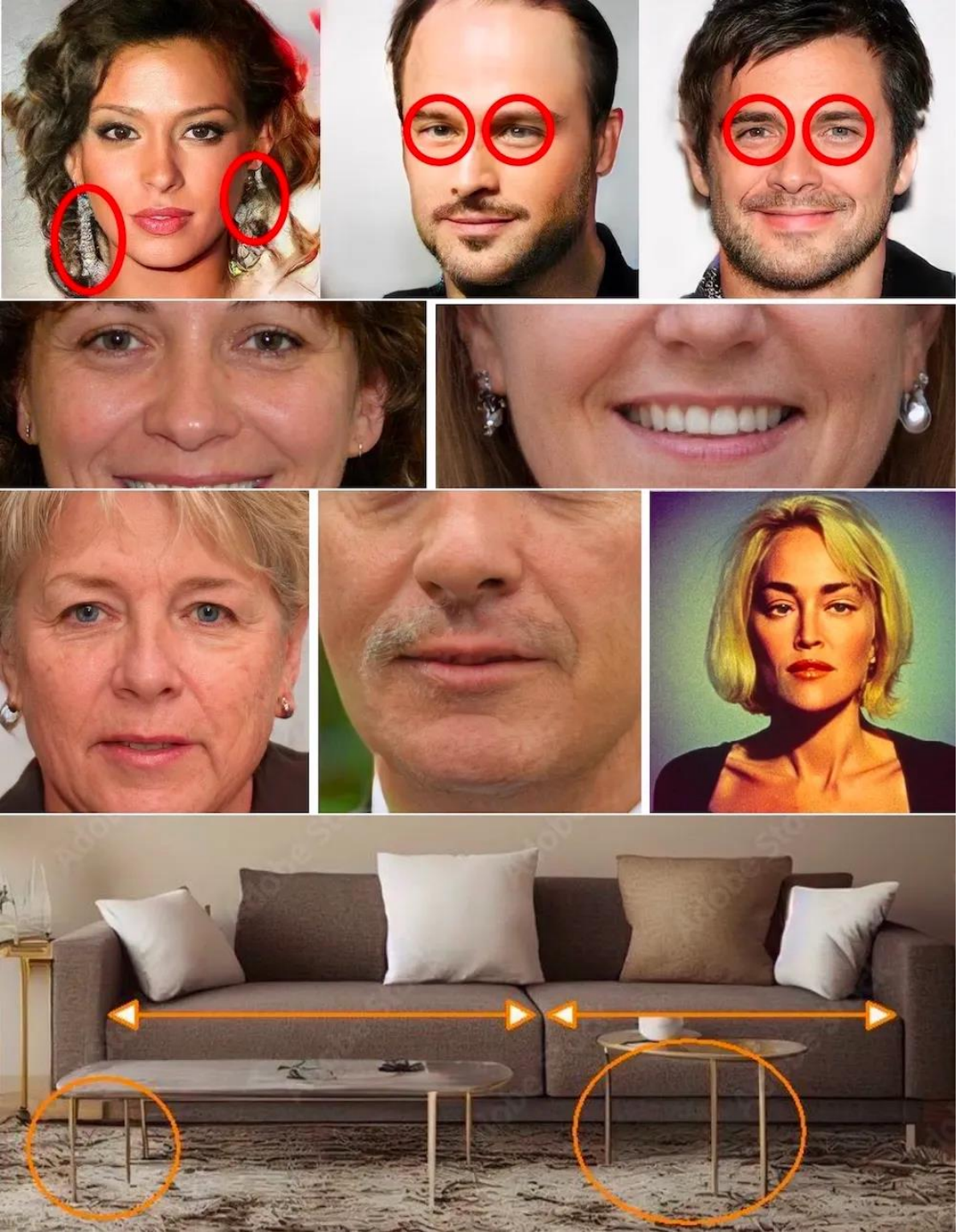
Declarative Programming

Mathematical modelling, also SQL, PyTorch

‘What’ must be computed, instead of ‘how’

A wrong model still returns a ‘solution’.

Fewer examples (more specialized libraries)



Wrong model, valid output

1. **It runs**
   Compiles. Solver returns a solution.

2. **Looks like a solution**
   A schedule, a cost, …

3. **Still the wrong problem**
   E.g. one missing constraint.
   You would not notice at first glance.

[Img: https://medium.com/@aliborji/qualitative-failures-of-image-generation-models-36c5ce9ecc53]

Many models for the same problem

Same problem, different viewpoints / encodings.
Runtime can differ by 1000×. Both can be ‘correct’.

Viewpoint

Interval vs assignment vs sequence, …

Encoding

Globals vs pairwise; MIP vs CP vs SAT; auxiliaries, …

Symmetry

Optional. Changes which solutions you get.

Correct ≠ good

A correct model can still be a bad model.

Modelling library vs modelling language

Python libraries

CPMpy, OR-Tools, Pyomo, Gurobi API, ...

Lots of Python on the web.

Domain-specific languages

MiniZinc, Essence, AMPL, ...

Much less on the web.

Why still use a solver?

If it can write Python, why not just let it write a search algorithm in Python?

1 Feasibility, proofs

Feasible, or UNSAT. Sometimes an optimality proof.

2 Generic

Can change the model, the data, the objectives…
Nothing is hardcoded for one problem variant

3 Years of engineering

When using a solver, vs freshly written code

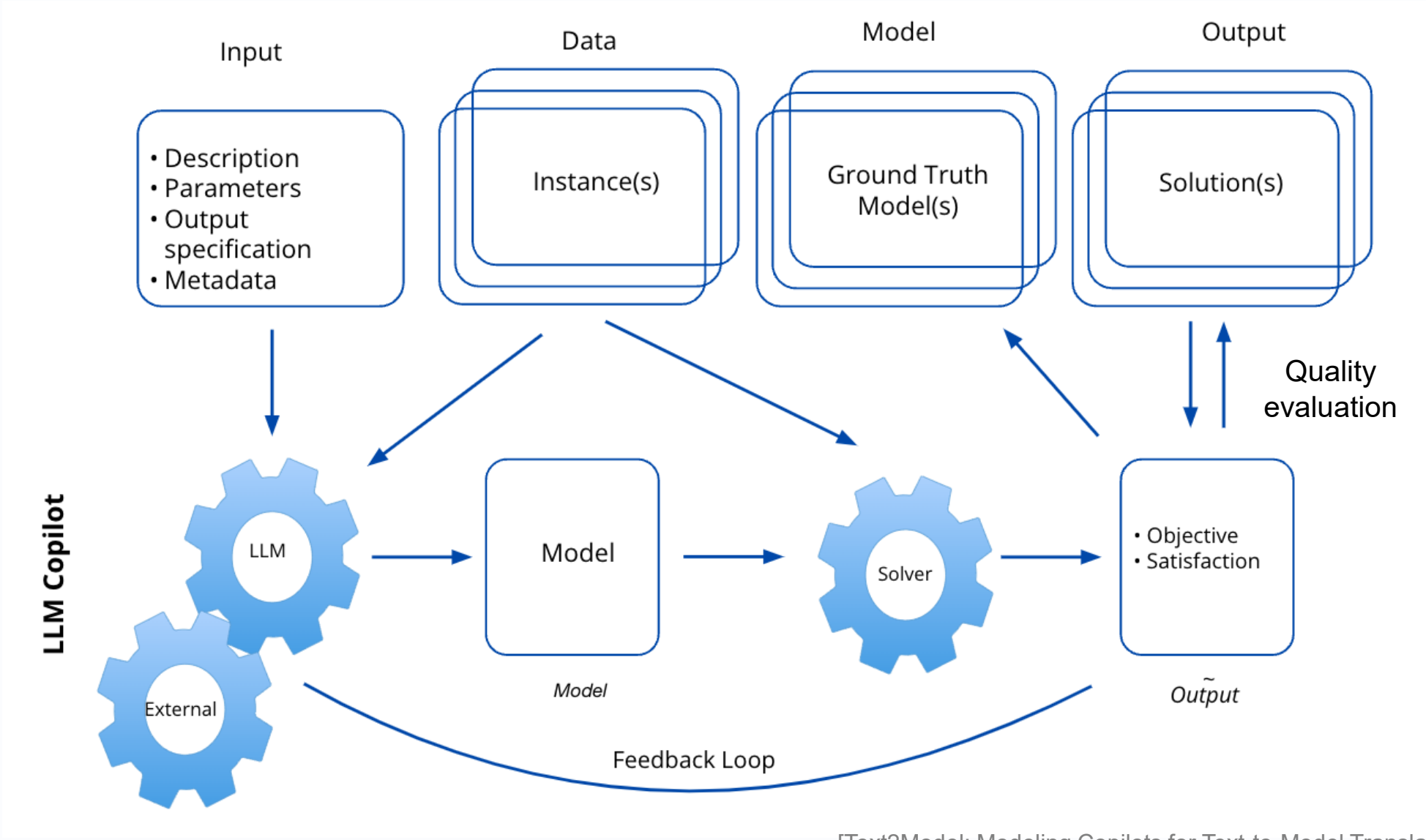
+ often parallel solving for free

4 Token-efficient

A model is much shorter and more readable than a search algorithm for a problem,
let alone then a text-based reasoning trace.

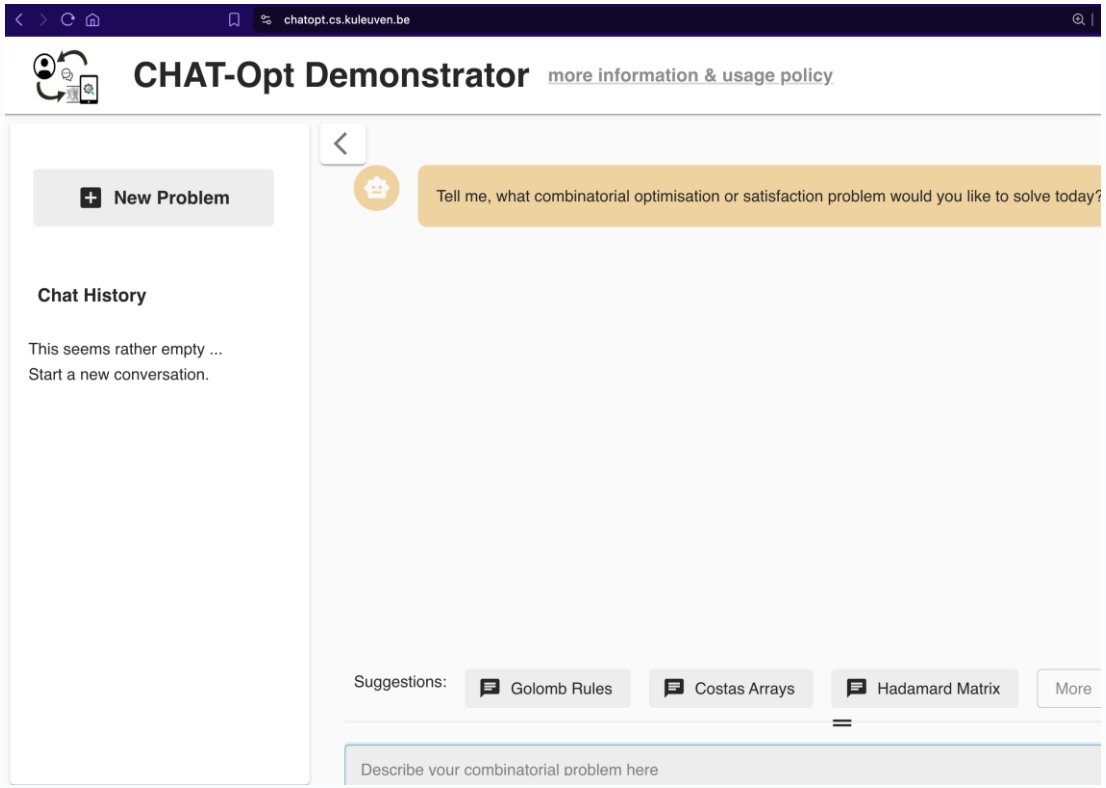
Formalize, Don’t Optimize: The Heuristic Trap in LLM-Generated Combinatorial Solvers, Wang et. al.

Modeling CoPilots



[Text2Model: Modeling Copilots for Text-to-Model Translation, S. Kadioglu et. al.]

Can Large Language Models model Constraint Problems?



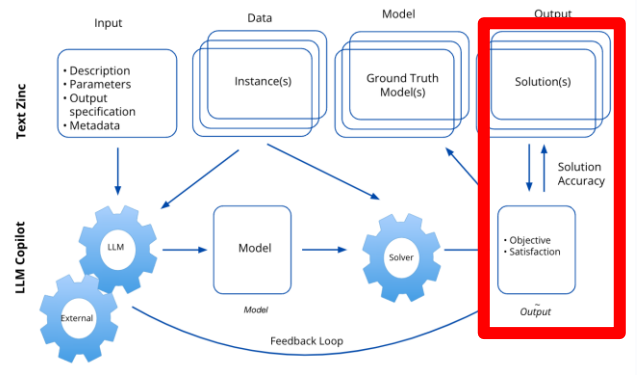
ChatOpt Online Demo

https://chatopt.cs.kuleuven.be

European Research Council
Established by the European Commission

Quality Evaluation

What counts as success?



1 Parses	Well-formed output.
2 Compiles	Accepted by the modelling language / API.
3 Solver returns something	A solution, or a clean UNSAT.
4 Right problem?	Feasible for the intended problem (soundness).
5 Complete / optimal	Nothing missing; UNSAT means UNSAT.
6 A good model	Also solves at a realistic size.

Model vs instance

The model

Parameterized: *n* employees, a demand table, a horizon, …

Works for any instance, in principle…

This is what modeling experts write.

```
def knapsack(values, weights, capacity):
    """Return a CPMpy model and decision variables for
 a 0/1 knapsack instance."""
    x = boolvar(shape=len(values), name="x")

    model = Model(sum(x * weights) <= capacity)
    model.maximize(sum(x * values))
    return model, x
```

One instance

Concrete numbers.

Agreeing on 3 instances ≠ the model is correct. Two models can agree on some, and differ on others.

```
# example instance
values = [10, 40, 30, 50]
weights = [5, 4, 6, 3]
capacity = 10

model, x = knapsack(values, weights, capacity)
model.solve()
```

Soundness, completeness, optimal value

Property	Condition	Meaning
Solution soundness	$S_M \subseteq S_B$	Every solution of M is a solution of the intended problem.
Solution completeness	$S_B \subseteq S_M$	Nothing feasible was lost. UNSAT of M ⇒ actually unsat.
Solution-set equivalence	$S_M = S_B$	Same solutions. Safest if later constraints may be added.
Optimal-value preservation	$S_M = S_B$ & $v(S_M) = v(S_B)$	Same solutions with same objective value.

How much data does a check use?

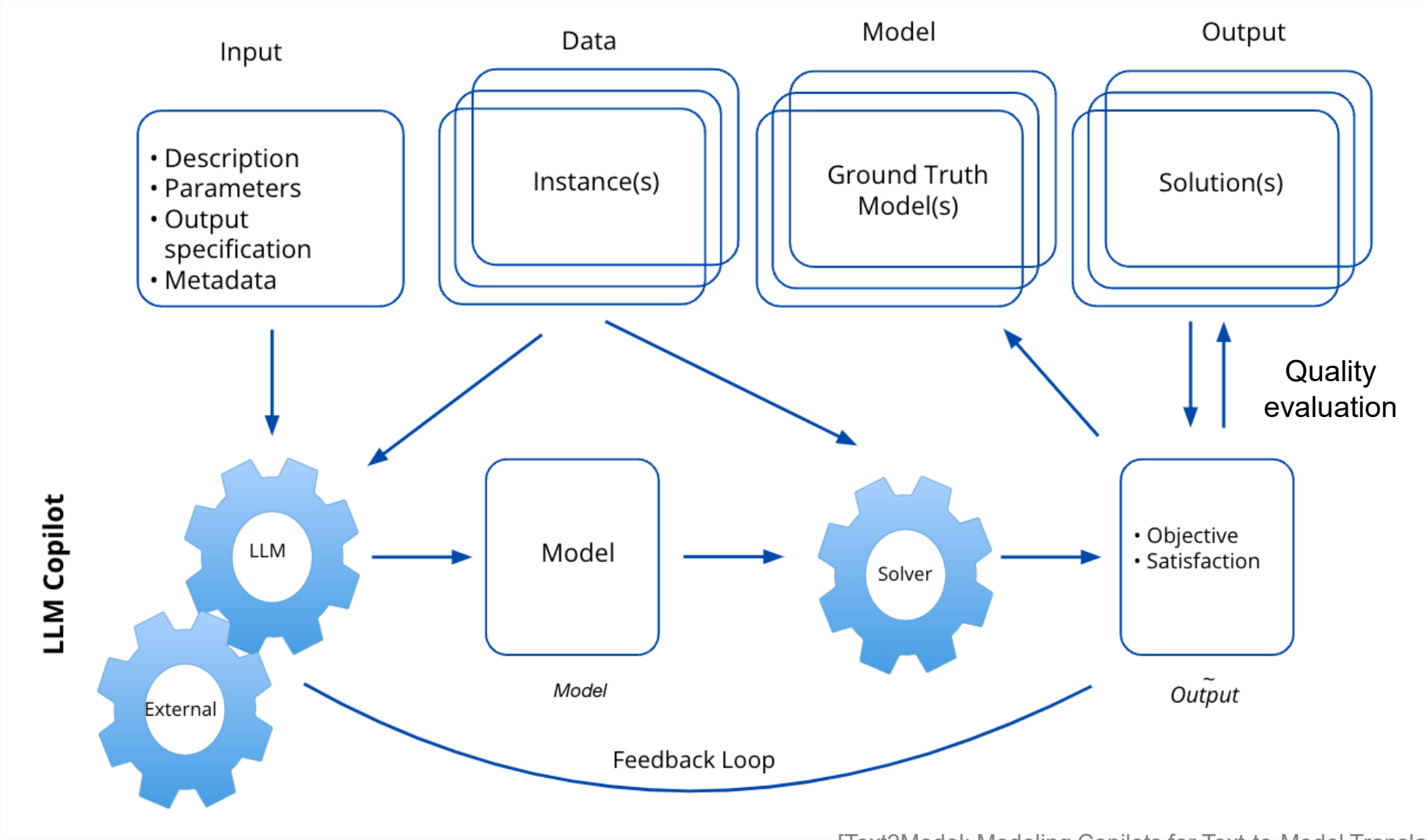
1. **One solution that must be returned**
   Fixed target, very limited...

2. **One instance**
   Like a unit test.

3. **A test set**
   Stronger statistical guarantees than a single sample; common in ML

4. **All possible instances**
   Undecidable in general; sometimes provable.

Modeling CoPilots



[Text2Model: Modeling Copilots for Text-to-Model Translation, S. Kadioglu et. al.]

Tutorial Agenda

Session A: Introduction, data, evaluation (9:00 - 10:30)

A1) Introduction (30 min): Tias Guns

A2) Benchmarking & Evaluation (60 min): Dimos Tsouros

Break (10:30 - 11:00)

Session B: Modelling copilots (11:00 - 12:30)

B1) Modelling copilots (60 min): Serdar Kadioglu

B2) Wrap-up (30 min): Tias Guns

Session A.2

Benchmarking & Evaluation

Presenter: Dimos Tsouros

LLMs for modeling constraint problems

- NL4Opt[1] NeurIPS competition
- Decomposition-based MILP synthesis[2]
- OptiMUS[3], ORLM[4] systems
- LLM-as-a-Judge[5]
- LLMOPT[6]
- SIRL[7]
- OptiChat[8]
- Text2Model[8]

And many many many more …..

1. Ramamonjison et al. “NL4Opt Competition: Formulating Optimization Problems Based on Their Natural Language Descriptions” (2022)
2. Li et al. “Synthesizing Mixed-Integer Linear Programming Models from Natural Language Descriptions” (2023)
3. AhmadiTeshnizi et al. “OptiMUS: Scalable Optimization Modeling with (MI)LP Solvers and Large Language Models” (2024)
4. Huang et al. “ORLM: A Customizable Framework in Training Large Models for Automated Optimization Modeling” (2025)
5. Frechette et al. “LLM-as-a-Judge for Combinatorial Optimization Modelisations from Natural Language” (2025)
6. Jiang et al. “Learning to define and solve general optimization problems from scratch” (2024)
7. Chen et al. “Solver-Informed RL: Grounding Large Language Models for Authentic Optimization Modeling” (2025)
8. Chen et al. “OptiChat: Bridging Optimization Models and Practitioners with Large Language Models” (2025)
9. Kadioglu et. al. “Text2Model: Modeling Copilots for Text-to-Model Translation (2025)

LLMs for modeling constraint problems

- Rise of LLMs
- Coding assistants (e.g. Cursor) excel in general coding tasks
- *What about modeling assistants?* [1]

How well ~~Can~~ Large Language Models model Constraint Problems?

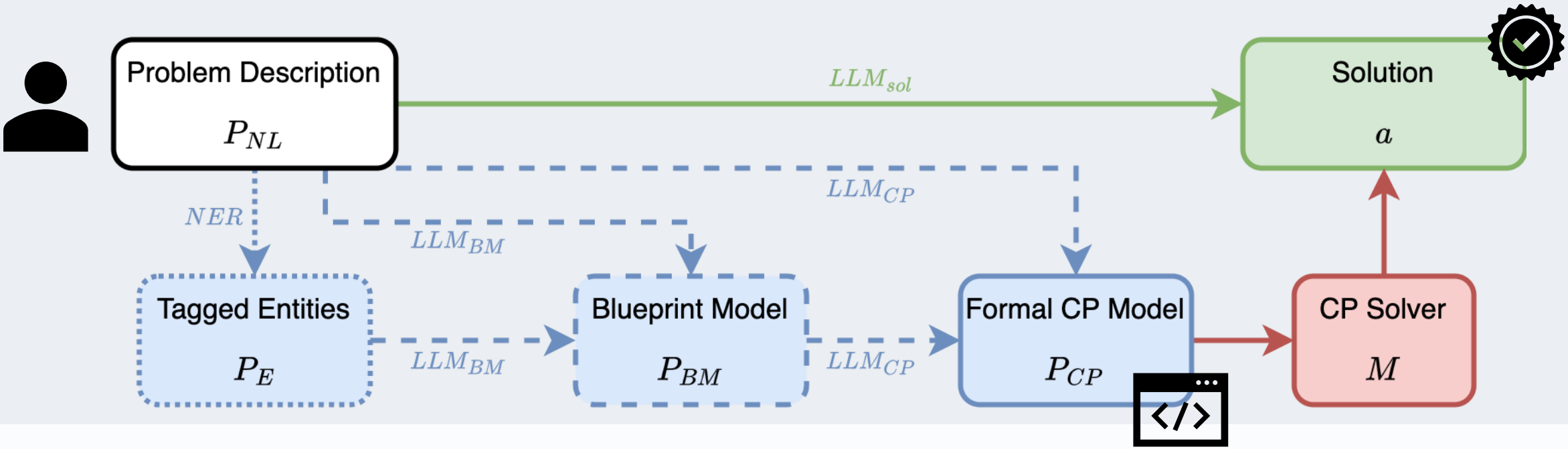
*Focusing on our experience on **challenges and insights** on benchmarking and evaluating LLMs for constraint modeling*

1. Tsouros, Verhaeghe, Kadioglu and Guns "Holy grail 2.0: From natural language to constraint models" (2023)

Evaluating LLMs for Constraint Modeling

Our first attempts – GPT 3.5 Era

Decomposing the problem to different subtasks



[1] Michailidis, Tsouros, and Guns. “Constraint Modelling with LLMs Using In-Context Learning” CP2024

In-Context Learning (ICL)

$$LLM(Prompt) = Response$$

In-Context Learning (ICL):
GPT 3.5 era!

- Task Learning:

$$LLM(C \oplus Q) \approx R$$

[1] Michailidis, Tsouros, and Guns. “Constraint Modelling with LLMs Using In-Context Learning” CP2024

Important question!

How should we evaluate our methods?

Which datasets?

Which evaluation metrics?

Easy, we got it

Experiments: Datasets

- **NL4Opt**[1]: Linear Optimization Problems
- **LGPs**[2]: Logic Grid Puzzles

} Homogeneous

- **Mixed**: 18 diverse problems drawn from a CP Modelling course.

} New

1 Ramamonjison et al. Nl4opt competition: 'Formulating optimization problems based on their natural language descriptions'. (2022)

2 Arindam Mitra and Chitta Baral. 'Learning to automatically solve logic grid puzzles'. (2015)

Evaluation Metrics

Declaration Accuracy

Model Accuracy

Solution Accuracy

```python
# Decision Variables
selected_movies = boolvar(shape=num_movies)  # 1 if the movie is selected, 0 otherwise

# CP Model
model = Model()

# Non-overlapping movie schedules
for i in range(num_movies):
    for j in range(num_movies):
        # Check if the intervals overlap for each pair of movies
        if i != j and movies[i][2] > movies[j][1] and movies[j][2] > movies[i][1]:
            # Constraint: the movies cannot be selected together
            model += ~(selected_movies[i] & selected_movies[j])

# Objective: Maximize the number of selected movies
model.maximize(sum(selected_movies))

# Solve
model.solve()
print(selected_movies.value())
# Output:
# [False False False False False  True  True  True  True]
```

#Errors: CP Model could not be run (Execution Accuracy)

Ramamonjison et al. Nl4opt competition: ‘Formulating optimization problems based on their natural language descriptions’. (2022)

Intermediate Representations

Configuration: gpt-3.5-turbo-0125, 4-shot static ICL.

Dataset	Method	#Err	Sol. (%)	Decl. (%)	Model (%)
NL4Opt	CP	7	81.31	87.81	79.24
	+ BM	8	84.43	89.93	82.01
	+ NER	8	85.47	88.60	80.62
LGPs	CP	11	57.00	80.45	55.00
	+ BM	18	58.00	70.69	58.00
	+ NER	20	54.00	67.77	50.00

Small Linear Opt. Problems

For more <u>complex</u> problems

Dataset	# ICL Shots	Sol (%)	Sol (%) + BM
Mixed	2	33.33%	16.67%
	4	50.00%	50.00%
	8	50.00%	44.44%
	12	55.56%	50.00%

Configuration: gpt-3.5-turbo-0125, static ICL, Solution Accuracy.

Evaluation challenge — lesson learned #1

```python
# Import libraries
from cpmpy import *
import json

n = 4
friends = Debra, Janet, Hugh, Rick = list(range(n))
friends_s = ["Debra", "Janet", "Hugh", "Rick"]

# foods[i] is the friend allergic to the ith food
eggs, mold, nuts, ragweed = foods = intvar(0, n - 1, shape=n, name="foods")
foods_s = ["eggs", "mold", "nuts", "ragweed"]

# surnames[i] is the friend with the ith surname
baxter, lemon, malone, fleet = surnames = intvar(0, n - 1, shape=n, name="surnames")
surnames_s = ["baxter", "lemon", "malone", "fleet"]
```

Ground-Truth

```python
# Decision Variables
# allergy[p] = allergy index (0=eggs,1=mold,2=nuts,3=ragweed) of person p (p: 0=Debra,1=Jan
allergy = cp.intvar(0, 3, shape=4, name="allergy")

# surname[p] = surname index (0=baxter,1=lemon,2=malone,3=fleet) of person p
surname = cp.intvar(0, 3, shape=4, name="surname")

# Inverse/channeling arrays:
# allergy_owner[a] = person who has allergy a
allergy_owner = cp.intvar(0, 3, shape=4, name="allergy_owner")
# surname_owner[s] = person who has surname s
surname_owner = cp.intvar(0, 3, shape=4, name="surname_owner")
```

Generated

- In general, evaluation needs **some form of mapping** between the decision variables.

Evaluation challenge — lesson learned #1

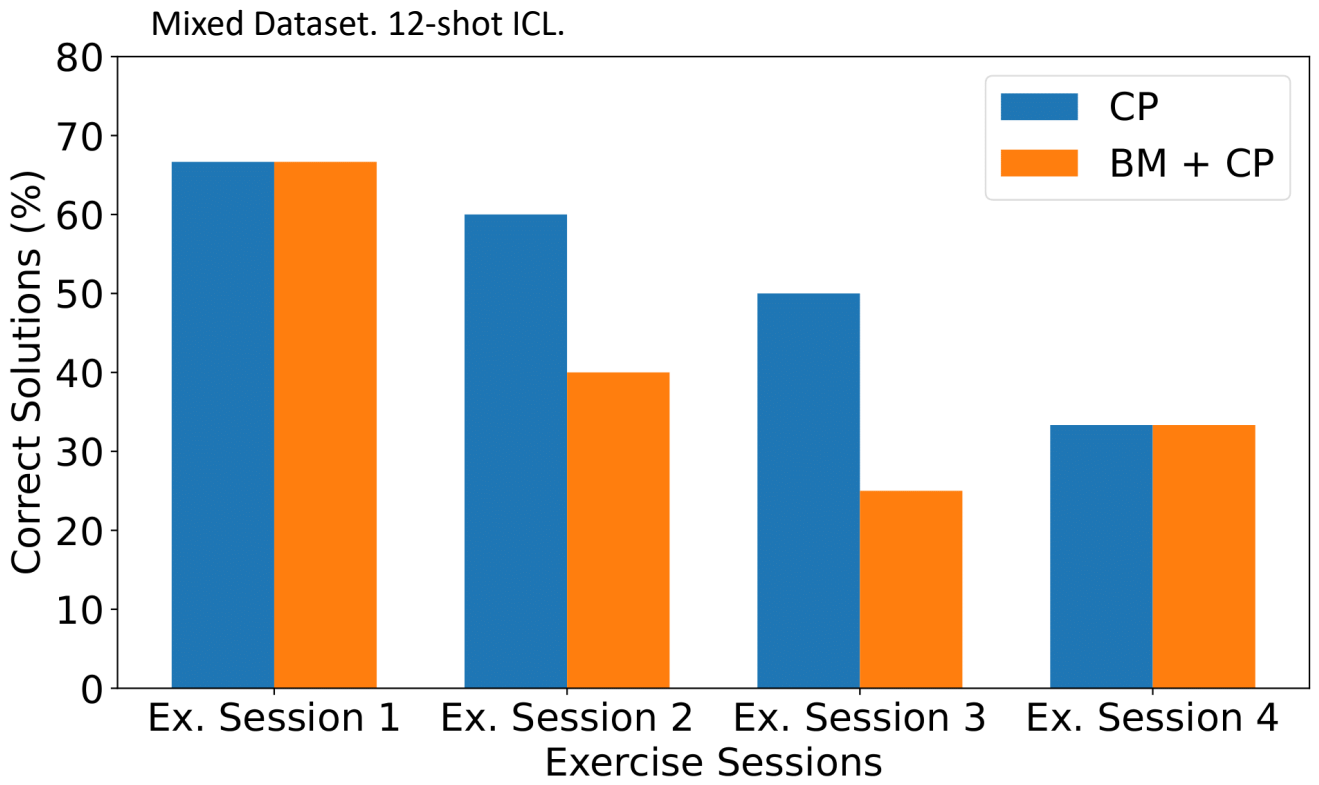
- In general, evaluation needs **some form of mapping** between the decision variables.

LLMs choice of Decision Variables is (and should be) free ⇒ How to evaluate Model Accuracy and Declaration Accuracy?
Automatic mapping worked for the homogeneous NL4Opt and LGP datasets, but **not** for bigger, more complex and diverse problems.

Focus on Solution Accuracy: Prompt LLMs to generate code that prints the solution in a specific format.

Solution accuracy drops on diverse problems

- Few similar examples to retrieve from.
- More complex problems.
- Lower accuracy.
- Below 40% by the fourth exercise.



Lesson learned #2: We need more complex and diverse problems for evaluation.

Benchmarking LLM-driven Constraint Modelling

CP-Bench

DCP-Bench-Open

Text2Zinc

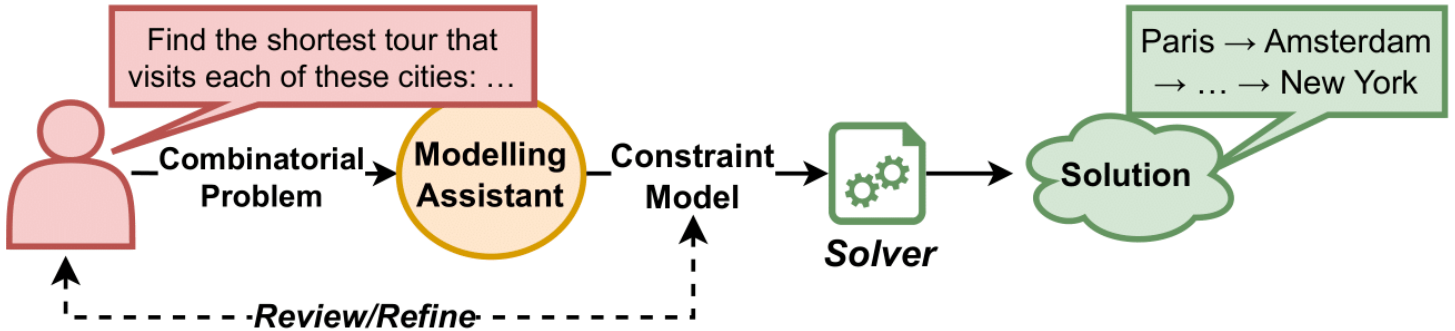
and challenges!

Existing LLM-based modelling datasets

- NL4Opt => nl4opt.github.io
- NLP4LP => huggingface.co/datasets/udell-lab/NLP4LP
- MAMO => huggingface.co/datasets/CardinalOperations/MAMO
- IndustryOR => huggingface.co/datasets/CardinalOperations/IndustryOR
- BWOR => huggingface.co/datasets/SJTU/BWOR
- ComplexOR => github.com/xzymustbexzy/Chain-of-Experts
- OptiBench => github.com/yangzhch6/ReSocratic
- CO-Bench => huggingface.co/datasets/CO-Bench/CO-Bench
- Text2Zinc => https://huggingface.co/datasets/skadio/text2zinc
- …

Most existing datasets focus is mostly on LP, MI(L)P problems, and only *optimization*
many include homogenous problems,
and are parallel efforts

Benchmarking LLM-based constraint modeling



- We need **representative benchmarks!**
- **Complex and Diverse**
- **Both satisfaction and optimization problems**
- **CP modeling (global constraints …)**

Dataset of problems/models

Evaluation Challenge: Multiple variable/modelling choices (viewpoints) & multiple (optimal) solutions
+ Evaluating **satisfiability of given solutions**

Benchmarking LLM-based constraint modeling

Our contributions:

- Focus on combining **both optimization & satisfaction** problems.
- **Solver agnostic** modeling frameworks
- **Clear separation** of problem description & instance data.

CP-Bench[1], DCP-Bench-Open[2]: Towards a larger, **diverse, combinatorial** problems dataset!

Text2Zinc[3]: Unified dataset to incorporate LP, MIP, and CP problems

1. Michailidis, Tsouros, and Guns. “CP-Bench: Evaluating Large Language Models for Constraint Modelling”, ECAI, (2025)
2. Michailidis, Tsouros, and Guns. “DCP-Bench-Open: Evaluating LLMs for Constraint Modelling of Discrete Combinatorial Problems” (2026)
3. Akash Singirikonda, Serdar Kadıoglu and Karthik Uppuluri. “Text2Zinc: A Cross-Domain Dataset for Modeling Optimization and Satisfaction Problems in MiniZinc” (2025)

Text2Zinc Statistics

1775

Total Problems

Natural language instances

11

Problem Domains

Aim for diverse application areas covered

Includes instances of mixed of **LP, MIPs, and CP problems** across **7 different sources**

Separate data from constraint model

Text2Zinc: Example Timetabling Problem – Description

```
  "description": "Lecture timings need to be scheduled for courses across a limited number of periods. Each
     course requires a specific number of lectures and can only be assigned to certain periods due to
     availability constraints. Some courses have conflicts due to having common students and cannot be scheduled
     at the same time. Additionally, there is a limited number of rooms that can be used and thus a maximum
     number of lectures that can occur simultaneously. How can we allocate lectures to periods while ensuring all
      constraints are met?",
  "identifier": "or_lp_ip_scheduling_problem_2",
  "metadata": {
    "name": "Timetable Problem", "domain": "Scheduling", "objective": "satisfy", "source": "hakank", "constraints
       ": [
     "forall", "<=", "+", "=", "sum"]
  }
}
```

An example input with description, parameters, metadata, and output fields.

Text2Zinc: Example Timetabling Problem – Model

model.mzn

```
include "globals.mzn";

% Input parameters
int: courses;
int: periods;
int: rooms;

array[1..courses, 1..periods] of int: available;
array[1..courses, 1..courses] of int: conflict;
array[1..courses] of int: requirement;

% Decision variables
array[1..courses, 1..periods] of var 0..1: timetable;
```

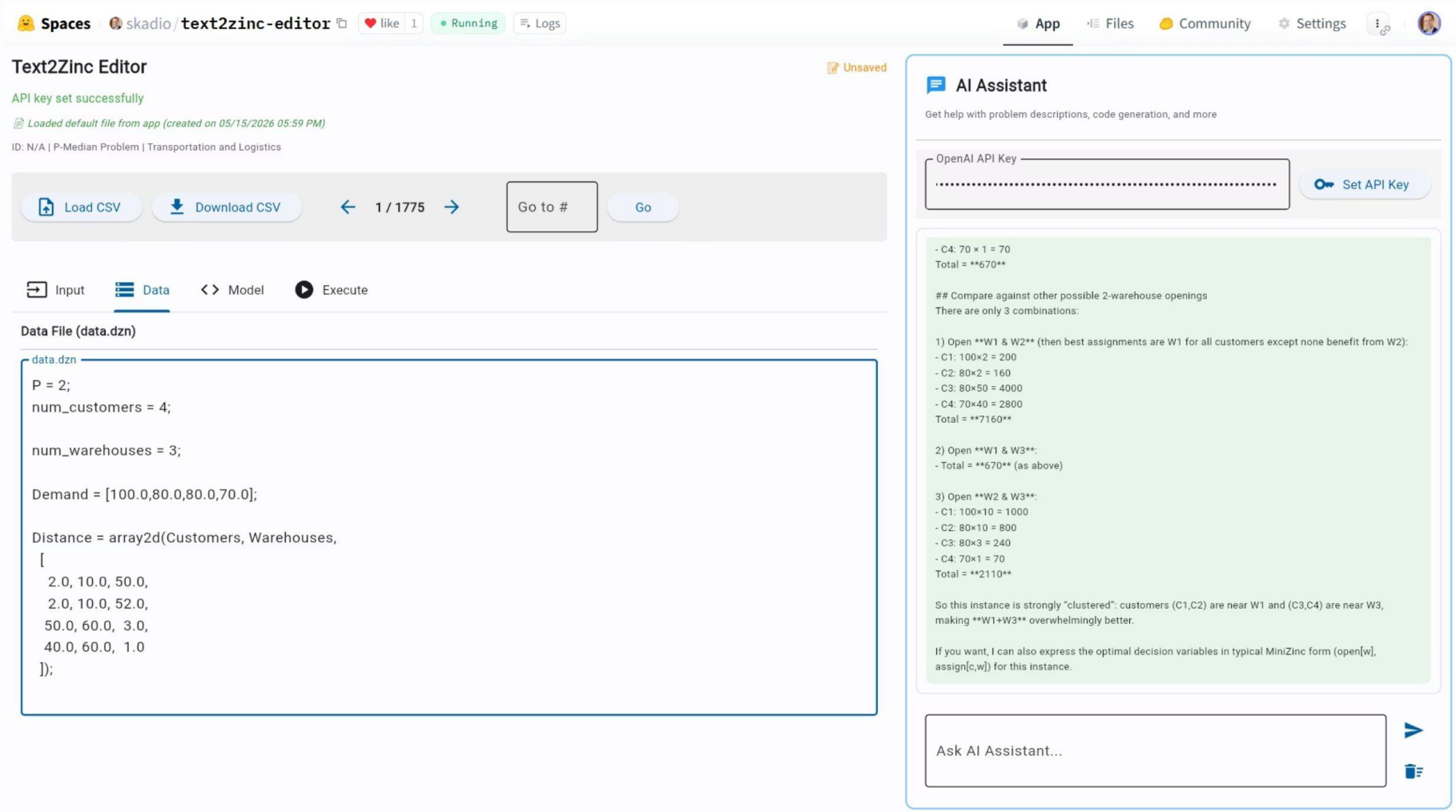
Text2Zinc: Example Timetabling Problem – data (separate)

data.dzn

```
int: courses = 5;
int: periods = 20;
int: rooms = 2;
array[1..courses, 1..periods] of int:
    available = array2d(1..courses,
   1..periods, [
   % 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6
      7 8 9 0
     0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
          1, 0, 1, 1, 0, 1, 1, 1, 1,
     1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1,
          1, 1, 1, 1, 1, 1, 1, 1, 1,
     0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1,
          1, 0, 1, 1, 1, 1, 0, 1, 1,
     1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1,
          1, 1, 1, 1, 1, 1, 0, 1, 1,
     1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
          1, 1, 1, 1, 1, 1, 1, 1, 1
```

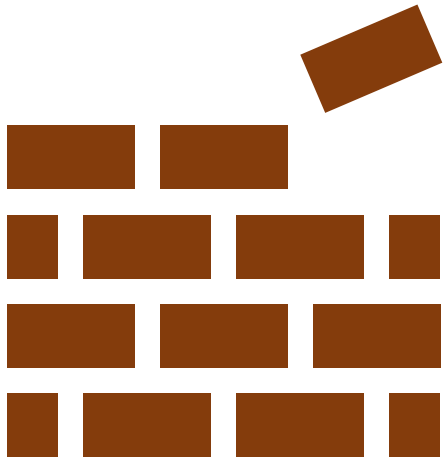
Text2Zinc Interactive Editor

https://skadio.github.io/text2model/#text2zinc



DCP-Bench-Open

- 164 *verified* & ***diverse*** instances
- Models in **CPMpy**
- Focus on **combinatorial problems**
- Evaluating **several frameworks** and modelling paradigms
- **Multi-instance evaluation** and more metrics



https://github.com/DCP-Bench/DCP-Bench-Open

DCP-Bench-Open: Example Car Sequencing problem– Description (as docstring)

DCP-Bench-Open / dataset / csplib_001_car_sequencing / **csplib_001_car_sequencing.cpmpy.py**

kostis-init initial commit

Code | Blame | 75 lines (63 loc) · 2.92 KB

```
1   #!/usr/bin/python3
2   # Category: csplib
3   # Source: https://github.com/CPMpy/cpmpy/blob/master/examples/car_sequencing.ipynb
4   # Source description: https://www.csplib.org/Problems/prob001/
5   # Problem instances: https://github.com/CPMpy/cpmpy/blob/master/examples/csplib/prob001_car_se
6
7   """
8   A number of cars are to be produced; they are not identical, because different
9   options are available as variants on the basic model.
10  The assembly line has different stations which install the various options
11  (air-conditioning, sunroof, etc.).
12  These stations have been designed to handle at most a certain percentage of
13   the cars passing along the assembly line.
14  Furthermore, the cars requiring a certain option must not be bunched together,
15  otherwise the station will not be able to cope.
16  Consequently, the cars must be arranged in a sequence so that the capacity of
17  each station is never exceeded.
18  For instance, if a particular station can only cope with at most half of the
19  cars passing along the line, the sequence must be built so that at most 1 car
20  in any 2 requires that option.
```


```
For instance, if a particular station can only cope with at most half of the
cars passing along the line, the sequence must be built so that at most 1 car
in any 2 requires that option.

Print the sequence of car types in the assembly line (sequence);
each car type is represented by a number starting from 0.
"""
```

DCP-Bench-Open: Example Car Sequencing problem– Model

```
# Parameters
n_cars = sum(demand)  # The amount of cars to sequence
n_options = len(at_most)  # The amount of different options
n_types = len(demand)  # The amount of different car types
requires = cpm_array(requires)  # For element constraint

# Decision Variables
sequence = intvar(0, n_types - 1, shape=n_cars, name="sequence")  # The sequence of car types
setup = boolvar(shape=(n_cars, n_options), name="setup")  # Sequence of different options based on the car type

# Model
model = Model()

# The amount of each type of car in the sequence has to be equal to the demand for that type
model += [sum(sequence == t) == demand[t] for t in range(n_types)]

# Make sure that the options in the setup table correspond to those of the car type
for s in range(n_cars):
    model += [setup[s, o] == requires[sequence[s], o] for o in range(n_options)]

# Check that no more than "at most" car options are used per "per_slots" slots
for o in range(n_options):
    for s in range(n_cars - per_slots[o] + 1):
        slot_range = range(s, s + per_slots[o])
        model += (sum(setup[slot_range, o]) <= at_most[o])
```

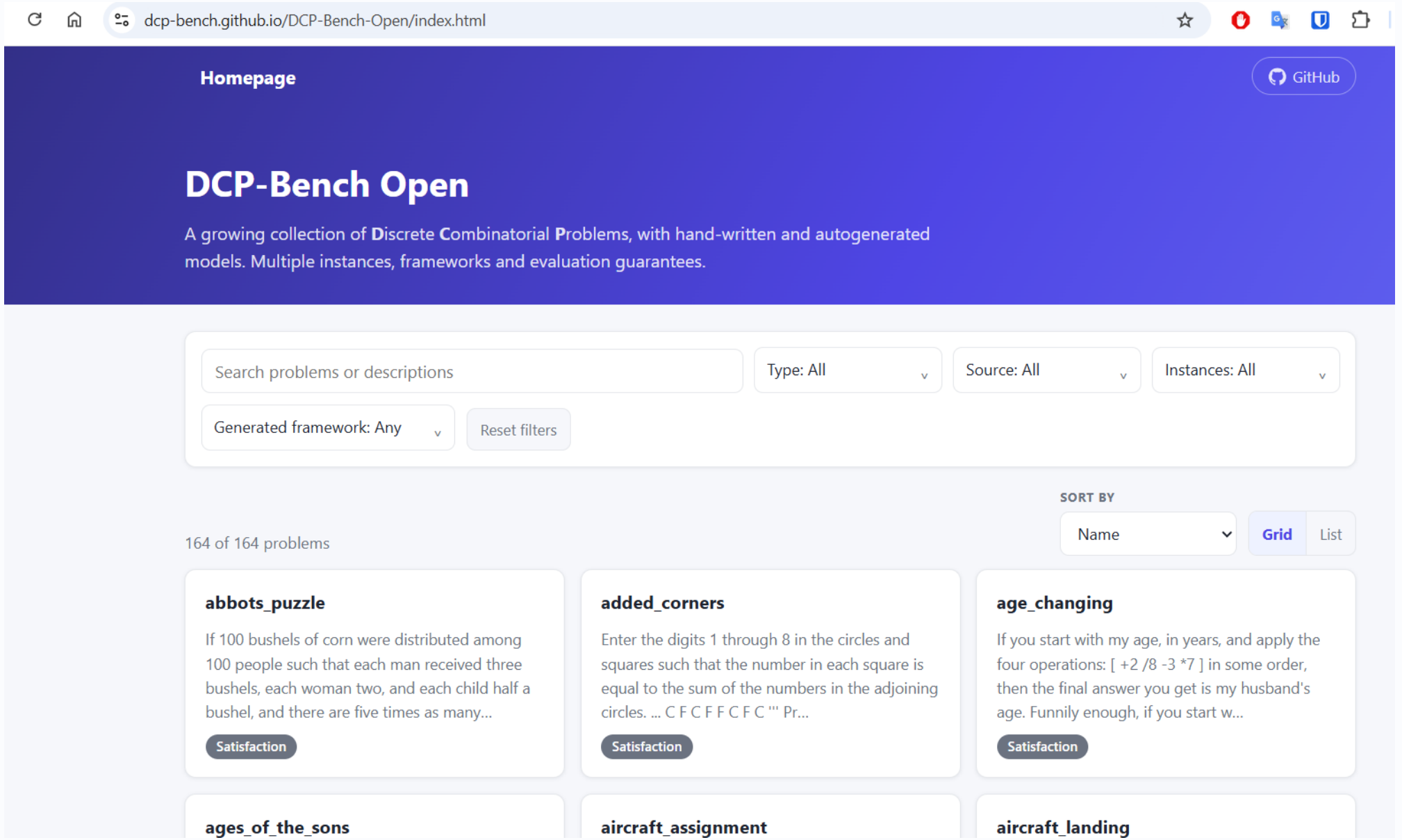
DCP-Bench-Open: Example Car Sequencing problem– data (separate)

DCP-Bench-Open / dataset / csplib_001_car_sequencing / csplib_001_car_sequencing.json

Code | Blame | 2418 lines (2418 loc) · 55.1 KB


```
    "at_most": [1, 2, 2, 2, 1],
    "per_slots": [2, 3, 3, 5, 5],
    "demand": [1, 1, 2, 2, 2, 2],
    "requires": [
      [1, 0, 1, 1, 0],
      [0, 0, 0, 1, 0],
      [0, 1, 0, 0, 1],
      [0, 1, 0, 1, 0],
      [1, 0, 1, 0, 0],
      [1, 1, 0, 0, 0]
    ]
  }
  ,
  {
    "at_most": [1, 2, 1, 2, 1],
    "per_slots": [2, 3, 3, 5, 5],
    "demand": [6, 10, 2, 2, 8, 15, 1, 5, 2, 3, 2, 1, 8, 3, 10, 4, 4, 2, 4, 6, 1, 1],
    "requires": [
      [1, 0, 0, 1, 0],
      [1, 1, 1, 0, 0],
      [1, 1, 0, 0, 1],
      [0, 1, 1, 0, 0],
      [0, 0, 0, 1, 0],
      [0, 1, 0, 0, 0],
      [0, 1, 1, 1, 0],
      [0, 0, 1, 1, 0],
      [1, 0, 1, 1, 0],
      [0, 0, 1, 0, 0]
```

DCP-Bench-Open: Example Car Sequencing problem– data (separate)



General-purpose benchmark datasets.*

Datasets	Scales	Problem Types	Year
NL4Opt[1]	245	LP	2022
NLP4LP[2]	67	LP, MILP	2024
NL2OPT[3]	70	LP, MILP, QP	2024
ComplexOR[4]	37	MILP	2024
MAMO[5]	1,209	LP, MILP, ODE (Easy & Complex)	2025
IndustryOR[6]	100	LP, IP, MILP, NLP	2025
OptiBench[7]	605	LP, IP, MILP, NLP	2025
DP-BENCH[8]	132	DP	2025
Bench4Opt[11]	394	LP, MILP	2025
OptMATH-Bench[12]	165	LP, IP, MILP, NLP, SOCP	2025
CP-BENCH[9]	101	CP	2025
CPEval[14]	25	CP	2025
CP-SynC-XL[15]	100	CP	2026
Text2Zinc[10]	1775	LP, MILP, CP	2025
DCP-Bench-Open[13]	164	CP	2026

** Most info from Xiu, Li, Fan, and Liu, "Large Language Models for Operations Research: A Comprehensive Survey," arXiv:2605.20849 (2026), Table 2.*

1. Ramamonjison et al. 'NL4Opt Competition: Formulating Optimization Problems Based on Their Natural Language Descriptions' (2022)
2. AhmadiTeshnizi et al. 'OptiMUS: Scalable Optimization Modeling with (MI)LP Solvers and Large Language Models' (2024)
3. Mostajabdaveh et al. 'Optimization Modeling and Verification from Problem Specifications Using a Multi-Agent Multi-Stage LLM Framework' (2024)
4. Xiao et al. 'Chain-of-Experts: When LLMs Meet Complex Operations Research Problems' (2024)
5. Huang et al. 'LLMs for Mathematical Modeling: Towards Bridging the Gap Between Natural and Mathematical Languages' (2025)
6. Huang et al. 'ORLM: A Customizable Framework in Training Large Models for Automated Optimization Modeling' (2025)
7. Yang et al. 'OptiBench Meets ReSocratic: Measure and Improve LLMs for Optimization Modeling' (2025)
8. Zhou et al. 'Auto-Formulating Dynamic Programming Problems with Large Language Models' (2025)
9. Michailidis et al. 'CP-Bench: Evaluating Large Language Models for Constraint Modelling' (2025)
10. Singirikonda et al. 'Text2Zinc: A Cross-Domain Dataset for Modeling Optimization and Satisfaction Problems in MiniZinc' (2025)
11. Wang et al. 'ORGEval: Graph-Theoretic Evaluation of LLMs in Optimization Modeling' (2025)
12. Lu et al. 'OptMATH: A Scalable Bidirectional Data Synthesis Framework for LLM-Based Optimization Modeling' (2025)
13. Michailidis, Tsouros, and Guns. 'DCP-Bench-Open: Evaluating LLMs for Constraint Modelling of Discrete Combinatorial Problems' (2026)
14. Song and Cohen. 'Do LLMs Understand Constraint Programming? Zero-Shot Constraint Programming Model Generation Using LLMs' (2025)
15. Wang et al. 'Formalize, Don't Optimize: The Heuristic Trap in LLM-Generated Combinatorial Solvers' (2026)

Task-specific benchmark datasets.*

Datasets	Domains	Year
NLGraph[1]	29,370 graph-structured optimization problems covering 8 task types	2023
PPNL[2]	Path planning tasks	2024
asyncHow[3]	Problem instances from domains such as diet, education, and healthcare	2024
RoutBench[4]	Including 1,000 VRP instances and their variants	2025
ALE-BENCH[5]	NP-hard problems such as path planning and production scheduling	2025
GraphArena[6]	4 polynomial-time challenges and 6 NP-hard optimization tasks	2025
CO-BENCH[7]	36 types of real-world combinatorial optimization tasks	2025
FrontierCO[8]	8 representative combinatorial optimization tasks	2025
OPT-BENCH[9]	20 real-world ML tasks and 10 NP-hard combinatorial optimization problems	2025
HeuriGym[10]	9 domains including chip design, protein engineering, and electronic circuits	2026

1. Wang et al. “Can Language Models Solve Graph Problems in Natural Language?” (2023)
2. Aghzal et al. “Can Large Language Models Be Good Path Planners? A Benchmark and Investigation on Spatial-Temporal Reasoning” (2024)
3. Lin et al. “Graph-Enhanced Large Language Models in Asynchronous Plan Reasoning” (2024)
4. Li et al. “ARS: Automatic Routing Solver with Large Language Models” (2025)
5. Imajuku et al. “ALE-Bench: A Benchmark for Long-Horizon Objective-Driven Algorithm Engineering” (2025)
6. Tang et al. “GraphArena: Evaluating and Exploring Large Language Models on Graph Computation” (2025)
7. Sun et al. “CO-Bench: Benchmarking Language Model Agents in Algorithm Search for Combinatorial Optimization” (2025)
8. Feng et al. “A Comprehensive Evaluation of Contemporary ML-Based Solvers for Combinatorial Optimization” (2025)
9. Li et al. “OPT-BENCH: Evaluating LLM Agent on Large-Scale Search Spaces Optimization Problems” (2025)
10. Chen et al. “HeuriGym: An Agentic Benchmark for LLM-Crafted Heuristics in Combinatorial Optimization” (2026)

** Info from Xiu, Li, Fan, and Liu, “Large Language Models for Operations Research: A Comprehensive Survey,” arXiv:2605.20849 (2026), Table 4.*

Benchmarking LLMs for constraint modeling: Challenges

1. Evaluating the generated models! - from lesson learned #1
2. Selecting diverse problems (e.g. no duplicate models that differ only in param values) - from lesson learned #2

Challenge 1: Evaluating the generated models

How can we conclude that a model is correct?

Ground-Truth CP Model M_T

???

Generated CP Model M_G

Challenge 1: Evaluating the generated models

```python
# Parameters
n = 2  # Number of pack sizes
packs = [7, 13]  # Pack sizes
max_val = target * 2  # Arbitrary max limit of pack counts

# Decision variables
counts = intvar( lb: 0, max_val, shape=n, name="counts")  # C
total = intvar( lb: 0, max_val * n, name="total")  # Total nu

# Model
model = Model([
    total == (counts * packs).sum(),
    total >= target,
])

# Objective
model.minimize(total)

# Solve the model
model.solve()
```

Ground-Truth

```python
# Decision Variables
# Number of 7-packs and 13-packs
counts = cp.intvar( lb: 0, target, shape=2, name="counts")

# Constraints
# The total number of beers must be at least the target
model += counts[0]*7 + counts[1]*13 >= target
# Objective: minimize the total number of beers minus th
total_beers = counts[0]*7 + counts[1]*13
model.minimize(total_beers - target)

# Solve and print
if model.solve():
    solution = {'counts': counts.value().tolist()}
```

Generated

Challenge 1: Evaluating the generated models

```python
# Import libraries
from cpmpy import *
import json

n = 4
friends = Debra, Janet, Hugh, Rick = list(range(n))
friends_s = ["Debra", "Janet", "Hugh", "Rick"]

# foods[i] is the friend allergic to the ith food
eggs, mold, nuts, ragweed = foods = intvar(0, n - 1, shape=n, name="foods")
foods_s = ["eggs", "mold", "nuts", "ragweed"]

# surnames[i] is the friend with the ith surname
baxter, lemon, malone, fleet = surnames = intvar(0, n - 1, shape=n, name="surnames")
surnames_s = ["baxter", "lemon", "malone", "fleet"]
```

Ground-Truth

```python
# Decision Variables
# allergy[p] = allergy index (0=eggs,1=mold,2=nuts,3=ragweed) of person p (p: 0=Debra,1=Jan
allergy = cp.intvar(0, 3, shape=4, name="allergy")

# surname[p] = surname index (0=baxter,1=lemon,2=malone,3=fleet) of person p
surname = cp.intvar(0, 3, shape=4, name="surname")

# Inverse/channeling arrays:
# allergy_owner[a] = person who has allergy a
allergy_owner = cp.intvar(0, 3, shape=4, name="allergy_owner")
# surname_owner[s] = person who has surname s
surname_owner = cp.intvar(0, 3, shape=4, name="surname_owner")
```

Generated

Challenge 1: Evaluating the generated models

Model Equivalence[1]: $(\boldsymbol{M_T} \rightarrow \boldsymbol{M_G}) \wedge (\boldsymbol{M_G} \rightarrow \boldsymbol{M_T})$

Ideal, but it requires that Decision Variables are “connected” (channeling of M_G to M_T):

- What if M_G uses a **different viewpoint**?
- What if M_G is expressed in a **different framework**?
- Even our small CP Course-based dataset showed it is not straightforward!

[1] Michailidis, Kostis, Dimos Tsouros, and Tias Guns. "Constraint modelling with LLMs using in-context learning." (2024).

Challenge 1: Evaluating the generated models

- **The Problem:** How to automatically evaluate correctness?
- Evaluate the solution:
  1. The LLM generates a model.
  2. The model is executed to get a solution.
- (Multi-instance/Multiple) Objective value accuracy
- (Multi-instance/Multiple) Solution Accuracy
  1. Is solution valid and optimal against our ground-truth model?

$a_G \in Sol(C_T)$

$f(a_G) = f(a_T)$

Challenge 1: Evaluating the generated models

Solution accuracy vs Objective value accuracy

- Capturing correctness even **if the objective is modeled differently**
  - E.g. what if a linear objective is modeled with coefficients of different scale?
  - Then $f(a_G) \neq f(a_T)$
- Also evaluates **satisfiability of generated solution**
  - E.g. what if the generated model is underconstrained, and the given solution is not a solution of the GT?
  - We may have $f(a_G) = f(a_T)$, but not $a_G \in Sol(C_T)$
- Allows evaluating **satisfaction** problems

Challenge 1: Evaluating the generated models
Evaluation in CP-Bench (Example)

Printing Description: *Prompting with pre-specified key names for the solution printing!*

Problem Description:

```
"""
Given a target number of beers, find the closest combination of
 7-packs and 13-packs that meets or exceeds the target.

Print the counts of 7-packs and 13-packs used (counts).
"""
```

Generated Model:

```
# Decision Variables
# Number of 7-packs and 13-packs
counts = cp.intvar( lb: 0, target, shape=2, name="counts")

# Constraints
# The total number of beers must be at least the target
model += counts[0]*7 + counts[1]*13 >= target
# Objective: minimize the total number of beers minus the
total_beers = counts[0]*7 + counts[1]*13
model.minimize(total_beers - target)

# Solve and print
if model.solve():
    solution = {'counts': counts.value().tolist()}
    print(json.dumps(solution, indent=4))
```

Model can choose its dec vars, but has to follow the format -> the GT model has to already contain these as vars.

Michailidis, Kostis, Dimos Tsouros, and Tias Guns. "CP-Bench: Evaluating Large Language Models for Constraint Modelling“, ECAI, (2025)

Challenge 1: Evaluating the generated models

Evaluation in CP-Bench (Example)

```python
# Parameters
n = 2  # Number of pack sizes
packs = [7, 13]  # Pack sizes
max_val = target * 2  # Arbitrary max limit of pack counts

# Decision variables
counts = intvar( lb: 0, max_val, shape=n, name="counts")  # C
total = intvar( lb: 0, max_val * n, name="total")  # Total nu

# Model
model = Model([
    total == (counts * packs).sum(),
    total >= target,
])

# Objective
model.minimize(total)

# Solve the model
model.solve()
```

Ground-Truth

Adding the solution as a constraint – check if SAT + objective value

```python
# Decision Variables
# Number of 7-packs and 13-packs
counts = cp.intvar( lb: 0, target, shape=2, name="counts")

# Constraints
# The total number of beers must be at least the target
model += counts[0]*7 + counts[1]*13 >= target
# Objective: minimize the total number of beers minus th
total_beers = counts[0]*7 + counts[1]*13
model.minimize(total_beers - target)

# Solve and print
if model.solve():
    solution = {'counts': counts.value().tolist()}
    print(json.dumps(solution, indent=4))
```

Generated

Michailidis, Kostis, Dimos Tsouros, and Tias Guns. "CP-Bench: Evaluating Large Language Models for Constraint Modelling“, ECAI, (2025)

Challenge 1: Evaluating the generated models

*Importantly! For solution accuracy, we only need an (optimal) solution. Generated model can be in **any framework**!*

Gurobipy

Google OR-Tools

Michailidis, Kostis, Dimos Tsouros, and Tias Guns. "CP-Bench: Evaluating Large Language Models for Constraint Modelling“, ECAI, (2025)

Comparison across available general-purpose datasets

Dataset	Framework	Evaluation	Unit / multi-inst.
NL4Opt	Meaning representation / canonical form	Decl	Single
NLP4LP	Gurobi / PySCIPOpt	Exec · Obj	Single
NL2OPT	Problem Specification	Decl · Model	Single
ComplexOR	gurobipy + LP / PuLP	Exec · Obj	Single
MAMO	SciPy / SymPy; .lp → COPT	Exec · Obj	Single
IndustryOR	coptpy (ORLM) , framework agnostic	Exec · Obj	Single
OptiBench	PySCIPOpt	Exec · Obj	Single
DP-BENCH	DP code	Exec · Obj	Single
Bench4Opt	.lp	Model	**Multi**
OptMATH-Bench	gurobipy + LP/MPS; LLMOPT → Pyomo	Exec · Obj	Single
CP-Bench	CPMpy GT; allows evaluation for CPMpy / MZN / OR-Tools	**Exec · Sol · Obj**	Single
CPEVAL	MiniZinc + PyCSP3	**Exec · Sol · Obj**	**Multi**
CP-SynC	MiniZinc + PyCSP3; framework agnostic evaluation	**Exec · Sol · Obj**	**Multi**
Text2Zinc	MiniZinc	**Exec · Sol · Obj**	**Single; separate data to allow for multi**
DCP-Bench-Open	CPMpy GT; framework agnostic evaluation	**Exec · Sol · Obj**	**Multi**

Decl = declaration; Exec = execution; Obj = objective; Sol = solution; GT = ground truth;

Challenge 2: Selecting diverse problems

A lot of constraint model repositories exist, but…

E.g.: CSPLib, PyCSP3 examples, MiniZinc examples, Hakan's models, problem libraries and many more…

Can't we just use them as they are to evaluate LLMs on modelling?

No!

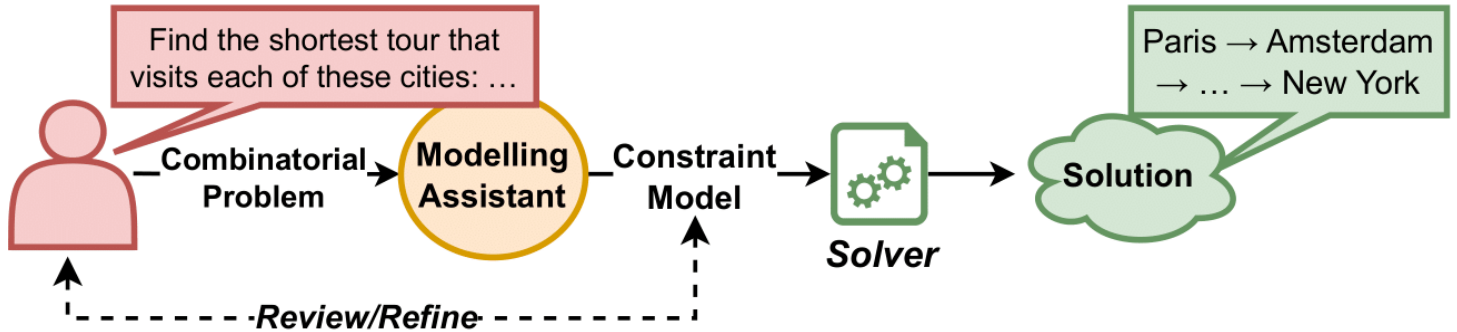
1. **Unstructured problem descriptions & models**
2. **Different formats**
3. **Different frameworks**
4. **Sometimes only model without description**
5. **Sometimes even only instance of model**

Challenge 2: Selecting diverse problems

- Use several sources of descriptions and models, and unify in a dataset:
  - **CSPLib** => csplib.org
  - **Hakan's repository** => hakank.org
  - **CPMpy repository** => github.com/CPMpy
  - …
- Text2Zinc: Unify several sources across different problem domains
- CP-Bench: Maintaining diversity and cover different problem types
  - E.g. Duplicate models that only differ in their parameter values

Michailidis, Kostis, Dimos Tsouros, and Tias Guns. "CP-Bench: Evaluating Large Language Models for Constraint Modelling“, ECAI, (2025)

LLMs as modelling assistants

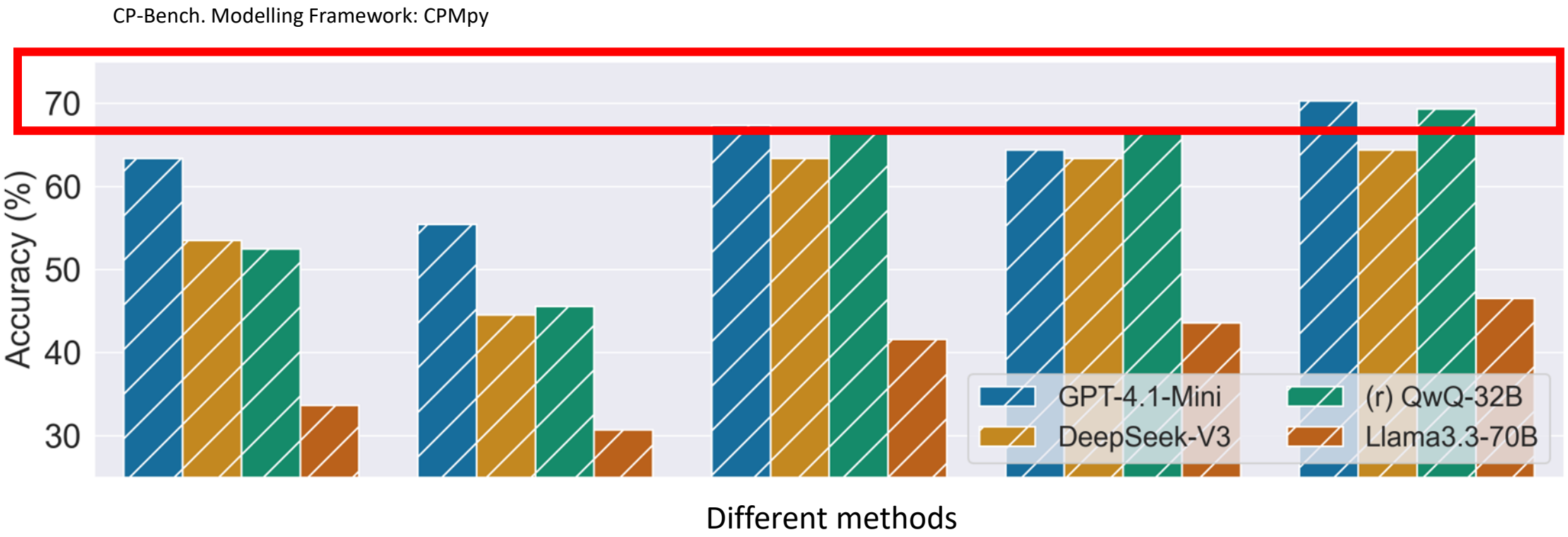


- We need **representative benchmarks!**
- **Complex and Diverse**
- **Satisfaction problems**
- **CP modeling (global constraints …)**

Dataset of problems/mod…

Evaluation Challenge: Multiple variable/modelling choices (viewpoints) & multiple (optimal) solutions
+ Evaluating **satisfiability of given solu…ns**

Some results on CP-Bench



Is 70% the ceiling?

No! More challenges than we thought

Michailidis, Kostis, Dimos Tsouros, and Tias Guns. "CP-Bench: Evaluating Large Language Models for Constraint Modelling“, ECAI, (2025)

Benchmarking LLMs for constraint modeling: Challenges

1. Evaluating the generated models! - from lesson learned #1
2. Selecting diverse problems (e.g. duplicate models that differ only in param values) - from lesson learned #2

Benchmarking LLMs for constraint modeling: Challenges

1. Evaluating the generated models! - from lesson learned #1
2. Selecting diverse problems (e.g. duplicate models that differ only in param values) - from lesson learned #2

(Existing sources containing descriptions and constraint models did not have in mind such an evaluation dataset)

3. Fixing “bad” problem descriptions (e.g. underspecified, ambiguous etc.) - Lesson learned #3

Michailidis, Kostis, Dimos Tsouros, and Tias Guns. "CP-Bench: Benchmarking LLMs for CP model generation", LLM-Solve Workshop, (2025)

Challenge: Problem(atic) descriptions

- Are underspecified, ambiguous: ***Lesson learned #3***
  - e.g., “With 3 dollars, you get 5 apples…”
  - Ambiguity: Can you also get 7 apples, or just 5, 10, 15, 20 etc. ?
- Contain irrelevant information
  - e.g. “A variant of this problem is to…”
- Contain limited information
  - e.g. “The problem is the Isomorphic Dice. Model it.”
- Provide modelling guidelines
  - e.g., “This problem can be modeled by…”

Michailidis, Kostis, Dimos Tsouros, and Tias Guns. "CP-Bench: Benchmarking LLMs for CP model generation", LLM-Solve Workshop, (2025)

Challenge: Problem(atic) descriptions

But still, the printing description (for solution accuracy evaluation) can be underspecified:

1. Indexing misalignment
   - 0-based vs. 1-based
2. Matrices orientation
   - Rows vs. Columns
3. Value types
   - 2D boolean array vs. 1D integer array

Michailidis, Kostis, Dimos Tsouros, and Tias Guns. "CP-Bench: Benchmarking LLMs for CP model generation", LLM-Solve Workshop, (2025)

Benchmarking LLMs for constraint modeling: Challenges

1. Evaluating the generated models! - from lesson learned #1
2. Selecting diverse problems (e.g. duplicate models that differ only in param values) - from lesson learned #2

(Existing sources containing descriptions and constraint models did not have in mind such an evaluation dataset)

3. Fixing “bad” problem descriptions (e.g. underspecified, ambiguous etc.) – from lesson learned #3
4. **Correct Ground-Truth models** (precisely matching the NL description) – Lessons learned #4-5

Michailidis, Kostis, Dimos Tsouros, and Tias Guns. "CP-Bench: Benchmarking LLMs for CP model generation", LLM-Solve Workshop, (2025)

Challenge: Ground-Truth Models are not perfect

Our goal is that each ground-truth model:

1. Precisely represents the problem description
2. Is self-contained (i.e. instantiated and executable)

Michailidis, Kostis, Dimos Tsouros, and Tias Guns. "CP-Bench: Benchmarking LLMs for CP model generation", LLM-Solve Workshop, (2025)

Challenge: Ground-Truth Models are not perfect

1. The model may make additional assumptions
   - e,g., “With 3 dollars you get 5 bananas”
   - Can you buy fruit individually or only in packs??

```
Model([the_sum == bananas+apples,
       # This don't work since "/" does integer division
       # 3*bananas/5 + 5*oranges/7 + 7*mangoes/9 + 9*apples/3 == 100
       # we multiply with 3*5*7*9=945 on both sides to weed out the
       3*bananas*189 + 5*oranges*135 + 7*mangoes*105 + 9*apples*315
       sum(x) == 100
       ])
```

Source: https://www.hakank.org/cpmpy/bananas.py

```
bananas_bundles = cp.intvar(1, 100, name="bananas_bundles")
oranges_bundles = cp.intvar(1, 100, name="oranges_bundles")
mangoes_bundles = cp.intvar(1, 100, name="mangoes_bundles")
apples_bundles = cp.intvar(1, 100, name="apples_bundles")

# Total fruits and total cost
total_fruits = (5 * bananas_bundles) + (7 * oranges_bundles)
total_cost = (3 * bananas_bundles) + (5 * oranges_bundles) +
```

An LLM-generated model

Lesson learned #4

Michailidis, Kostis, Dimos Tsouros, and Tias Guns. "CP-Bench: Benchmarking LLMs for CP model generation", LLM-Solve Workshop, (2025)

Challenge: Ground-Truth Models are not perfect

2. Include constraints not defined, e.g. symmetry-breaking constraints for faster solving (*but excluding feasible solutions…)*

```
# Symmetry breaking
# lexicographic ordering of rows
for r in range(N - 1):
    b = boolvar(N + 1)
    model += b[0] == 1
    model += b == ((matrix[r] <= matrix[r + 1]) &
                   ((matrix[r] < matrix[r + 1]) | b[1:] == 1))
    model += b[-1] == 0
```

Source: https://github.com/CPMpy/cpmpy/blob/master/examples/csplib/prob050_diamond_free.py

Lesson learned #5

Michailidis, Kostis, Dimos Tsouros, and Tias Guns. "CP-Bench: Benchmarking LLMs for CP model generation", LLM-Solve Workshop, (2025)

Challenge: Datasets' errors

Many (most) datasets for benchmarking LLMs for constraint modeling have faced similar issues:

Xiao et al., "A Survey of Optimization Modeling Meets LLMs: Progress and Future Directions", IJCAI 2025 made an analysis with human experts noticed similar issues in many known datasets:

Dataset	Size	Complexity	Error Rate
NL4Opt	289	5.59	$\geq 26.4\%$
IndustryOR	100	**14.06**	$\geq$ **54.0%**
EasyLP	652	7.12	$\geq 8.13\%$
ComplexLP	211	**13.35**	$\geq 23.7\%$
ReSocratic	605	7.45	$\geq 16.0\%$
NLP4LP	269	5.58	$\geq 21.7\%$
ComplexOR	37	5.98	$\geq 24.3\%$

Table 1: Quality statistics of optimization modeling benchmarks.

According to our results, we obtain several key findings. First, in current benchmarks, the error rate is relatively high. As shown in Table 1, except for EasyLP in MAMO, the error rates of other benchmarks exceed 15%, with IndustryOR even reaching as high as 54%, indicating that these benchmarks are not entirely reliable for evaluation. The errors can be caused by three main factors: (1) logical errors in problem descriptions, such as unbounded constraints; (2) poorly defined parameters that lead to unsolvable models; and (3) incorrect ground truth data. To address these issues, we manually filter all error cases and compile a unified, cleaned collection of optimization modeling benchmarks to facilitate future research.

Verified datasets

Verifying the dataset in-depth:

- Over-constrained Ground-Truth models (symmetry breaking)
- Solution printing under-specifications (indexing, value types, etc.)
- Modelling errors

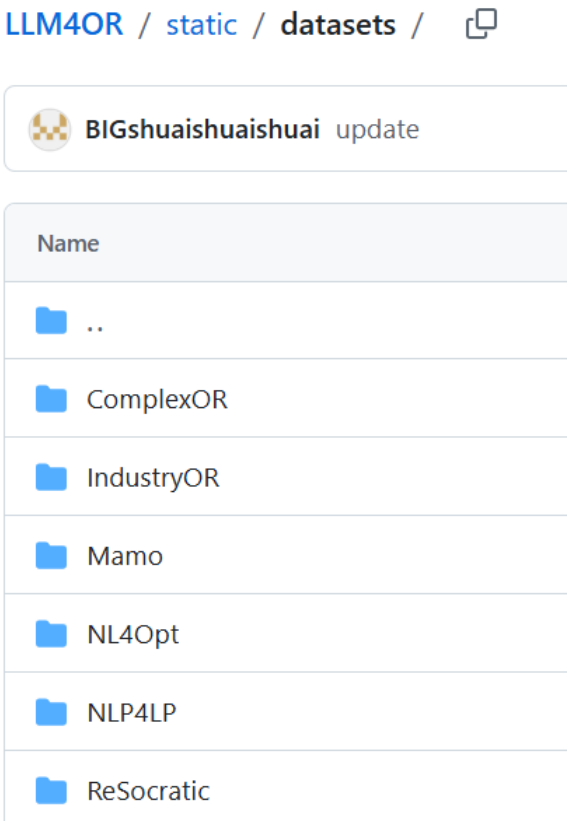
CP-Bench *Verified*: a subset of 63 verified problems

Text2Zinc verified: a subset of 110 verified problems

DCP-Bench-Open: Only verified - 164 verified problems

Verified datasets

Xiao et al., "A Survey of Optimization Modeling Meets LLMs: Progress and Future Directions", IJCAI 2025 also provides corrected versions of the following datasets:



https://github.com/LLM4OR/LLM4OR/tree/master/static/datasets

Differences of available datasets

Where do available datasets differ?

Main dimensions:

- Problems (and their diversity)
- Targeting satisfaction, optimization, or both types
- Modeling paradigm (CP, LP, MIP, …)
- Targeted modeling framework (gurobipy, pyomo, cpmpy, minizinc, …)
- Evaluation procedure and metrics
- Problem vs Instance (and support of multi-instance evaluation)
- Dataset verification

Differences of available datasets

Dataset	#	Scope	Paradigm	Framework	Evaluation	Unit / multi-inst.	Verified
NL4Opt	289 original; 245 HF; 230 LLMOPT	Opt.	LP	Meaning representation / canonical form	Decl	Single	Corrected (LLMOPT 230; SIRL 245)
NLP4LP	67 v1; 355 OptiMUS-0.3; 242 LLMOPT	Opt.	LP, MILP	Gurobi / PySCIPOpt	Exec · Obj	Single	Corrected (LLMOPT 242 feasible)
NL2OPT	70	Opt.	LP, MILP, QP	Problem Specification	Decl · Model	Single	**Yes: 70/70**
ComplexOR	37; 18 LLMOPT	Opt.	LP, MILP	gurobipy + LP / PuLP	Exec · Obj	Single	Corrected (LLMOPT 18/37 feasible)
MAMO	1,209	Opt. + ODE	LP, MILP, ODE	SciPy / SymPy; .lp → COPT	Exec · Obj	Single	Corrected (LLMOPT Easy relabel; SIRL Easy 642 / Complex 203)
IndustryOR	100	Opt.	LP, IP, MILP, NLP	coptpy (ORLM), framework agnostic	Exec · Obj	Single	Corrected (SIRL, later V2)
OptiBench	605	Opt.	LP, IP, MILP, NLP	PySCIPOpt	Exec · Obj	Single	**Yes: 605/605**
DP-BENCH	132	Opt.	DP	DP code	Exec · Obj	Single	**No**
Bench4Opt	394	Opt.	LP, MILP	.lp	Model	Multi	**No**
OptMATH-Bench	166	Opt.	LP, IP, MILP, NLP, SOCP	gurobipy + LP/MPS; LLMOPT → Pyomo	Exec · Obj	Single	**No**
CP-Bench	101	SAT + OPT	CP	CPMpy GT; allows evaluation for CPMpy / MZN / OR-Tools	Exec · Sol · Obj	Single	**63/101**
CPEVAL	30	SAT + OPT	CP	MiniZinc + PyCSP3	Exec · Sol · Obj	Multi	**Yes: 30/30**
CP-SynC	100	SAT + OPT	CP	MiniZinc + PyCSP3; · framework agnostic evaluation	Exec · Sol · Obj	Multi	**Yes: 100/100**
Text2Zinc	1,775	SAT + OPT	LP, IP, MILP, CP	MiniZinc	Exec · Sol · Obj	Single; separate data to allow for multi	**110/1775**
DCP-Bench-Open	164	SAT + OPT	CP	CPMpy GT; framework agnostic evaluation	Exec · Sol · Obj	Multi	**Yes: 164/164**

Decl = declaration; Exec = execution; Obj = objective; Sol = solution; HF = Hugging Face; GT = ground truth; MZN = MiniZinc.

Evaluating LLM-driven Constraint Modelling

So how well can LLMs model constraint problems?

Some insights from results in DCP-Bench-Open

Michailidis, Tsouros, and Guns. “DCP-Bench-Open: Evaluating LLMs for Constraint Modelling of Discrete Combinatorial Problems” (2026)

Constraint modeling instructions to LLMs

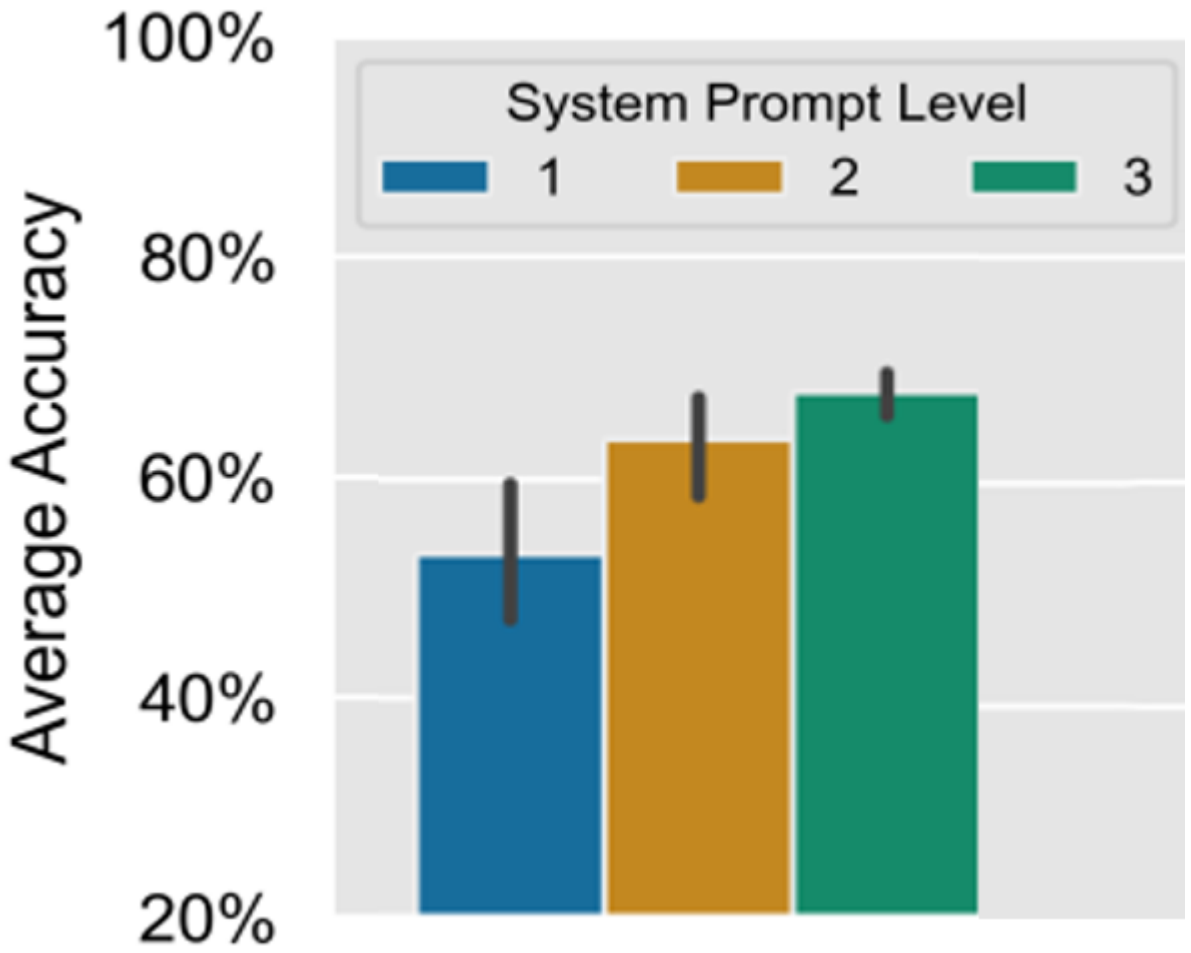
Evaluating effect of instructions in System Prompt.

3 Levels:

1. Basic prompt: Describe only the task
2. + Modeling guidelines: like guidelines to students
3. + Documentation of the modeling framework

Evaluating effect of instructions in System Prompt.

- Basic Prompt (1)
- Modelling Guidelines Prompt (2)
- Documentation Prompt (3)



Different datasets target different modeling frameworks

What type of modelling framework LLMs are better at?

Importantly! DCP-Bench-Open only needs an (optimal) solution. Generated model can be in any framework!

Google OR-Tools

Different datasets target different modeling frameworks

What type of modelling framework LLMs are better at?

- Python-based vs. Domain-Specific
- High-level vs. Low-level

Framework	Abstraction Level	Interface Type
MiniZinc	High	Domain-Specific
CPMpy	High	Python
OR-Tools	Medium	Python

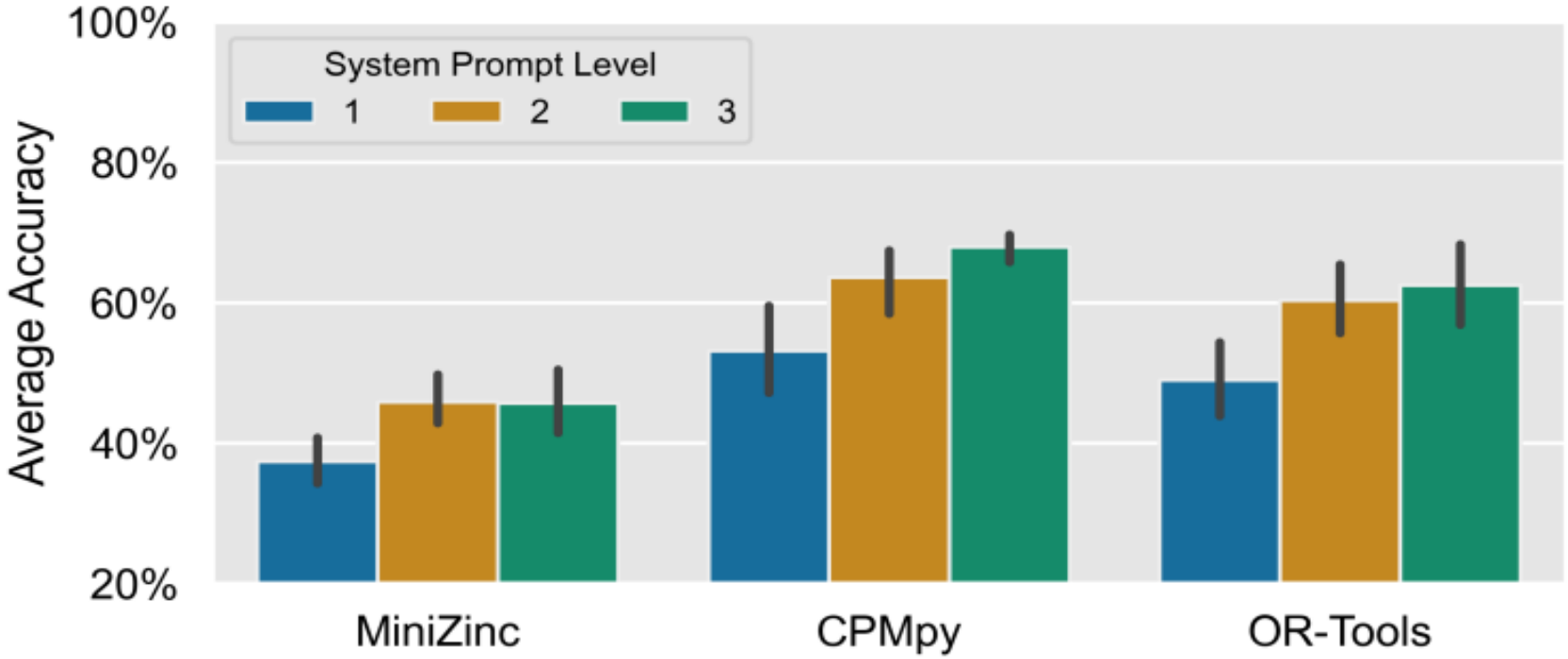
Google OR-Tools

Different datasets target different modeling frameworks

What type of modelling framework LLMs are better at?

- Python-based vs. Domain-Specific
- High-level vs. Low-level

Framework	Abstraction Level	Interface Type
MiniZinc	High	Domain-Specific
CPMpy	High	Python
OR-Tools	Medium	Python



Lesson learned #6:

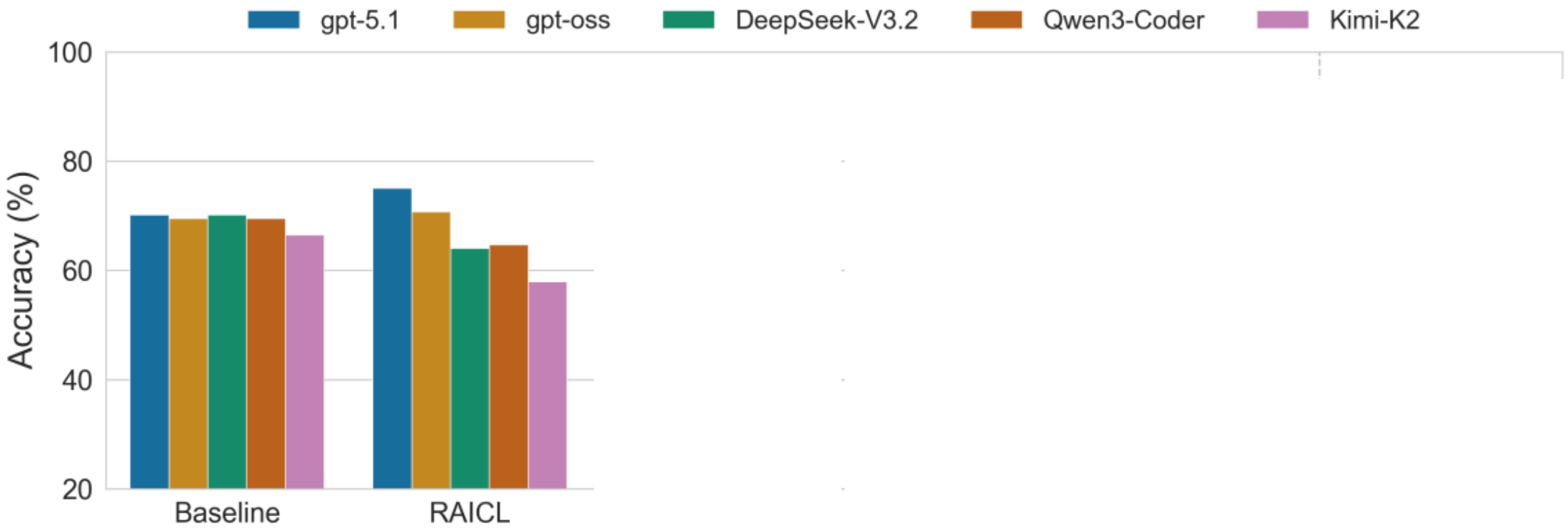
LLMs prefer **python**! (Other papers also had the same conclusion[1])

And LLMs prefer **high-level abstraction** …

1. Wang et al. “Formalize, Don't Optimize: The Heuristic Trap in LLM-Generated Combinatorial Solvers” (2026)

What about In-Context Learning?

- Results on DCP-Bench-Open (164 instances), CPMpy, system prompt 3

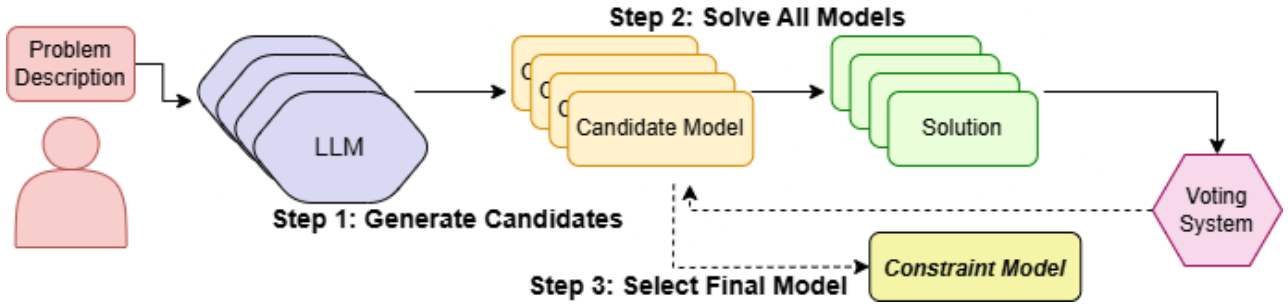


Lesson learned #7: In-context learning does not always help anymore.
Documentation prompt gives the important info + No homogenous problems

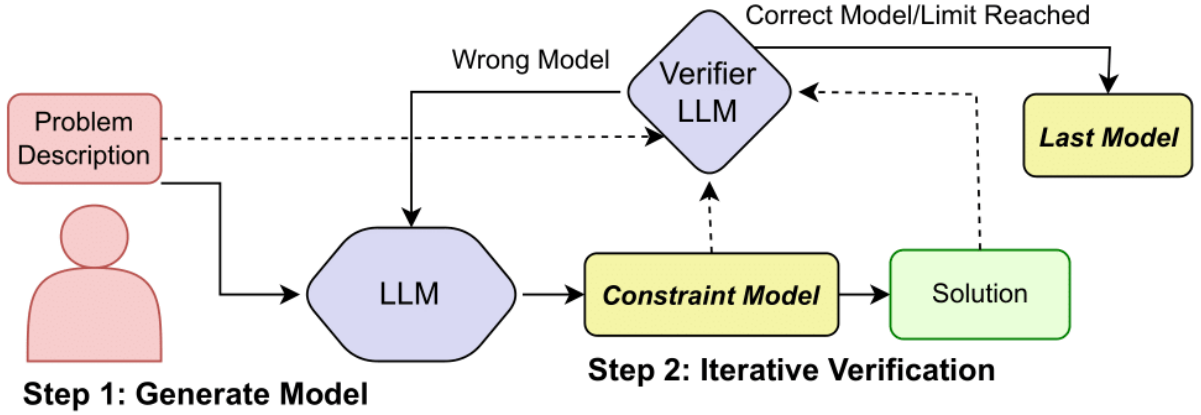
Inference-Time Compute Methods

Inspired from methods for coding assistants

1. Repeated Sampling



2. Self-Verification with tool calling



+ LLM internal reasoning

Evaluating Inference-Time Compute Methods

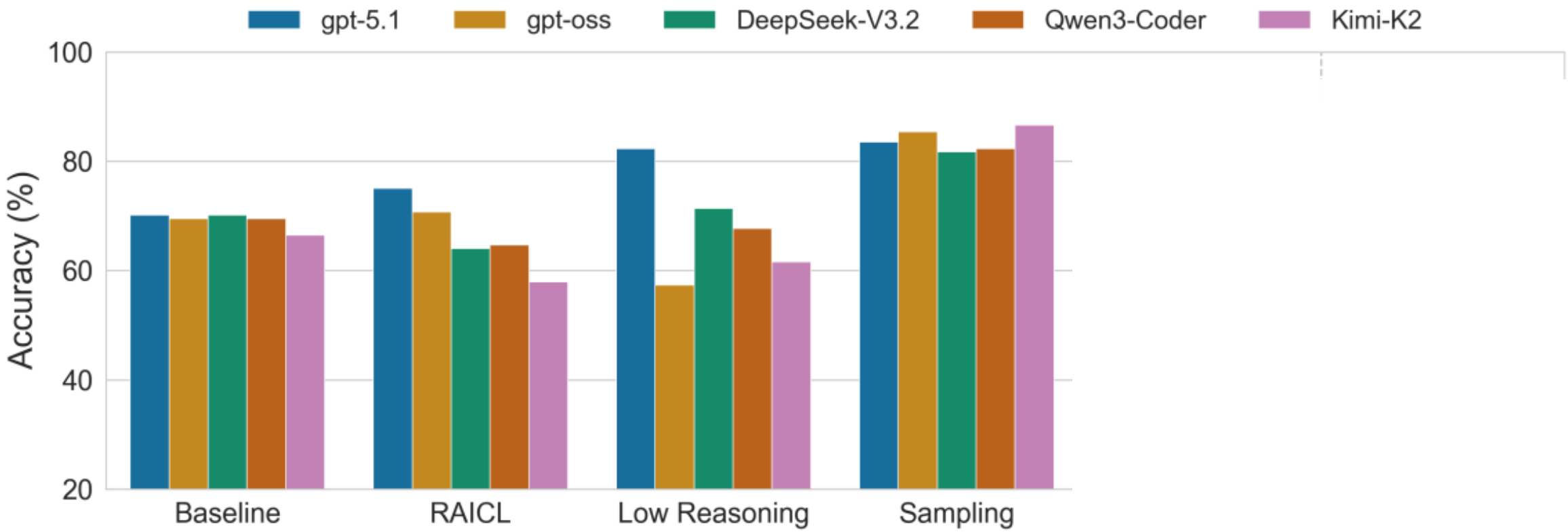
- Results on DCP-Bench-Open (164 instances), CPMpy, system prompt 3

Huge improvement in gpt-5.1, but lower than baseline in other LLMs.

- Generic **internal reasoning** not always effective.

Evaluating Inference-Time Compute Methods

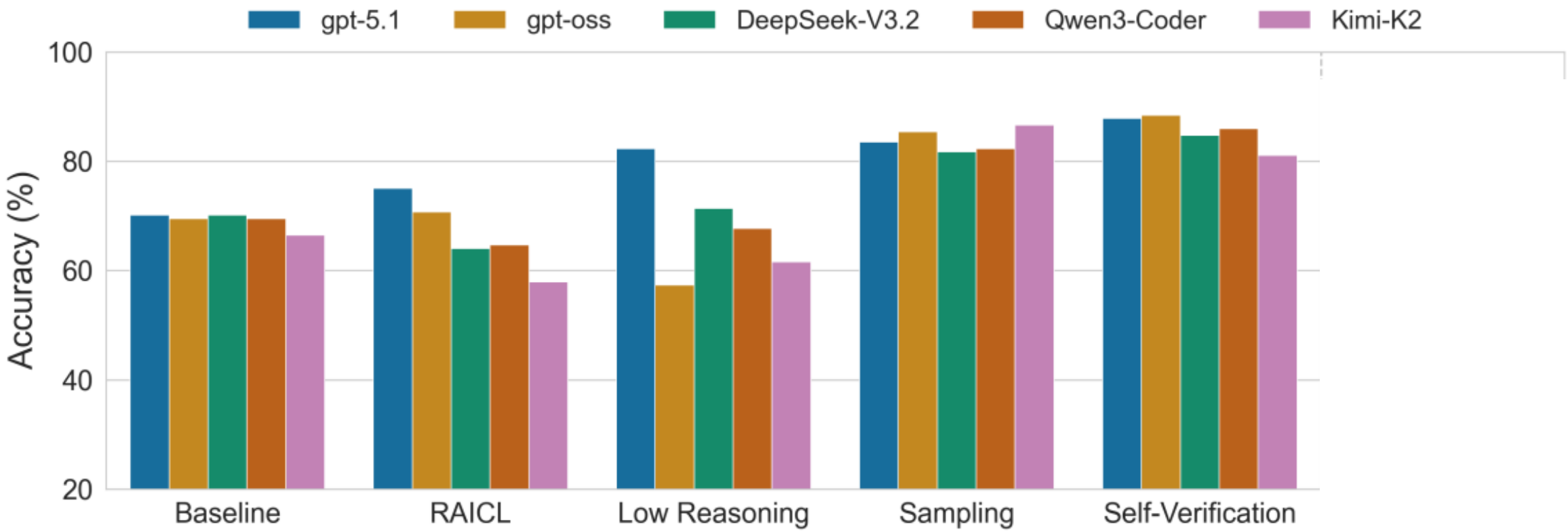
- Results on DCP-Bench-Open (164 instances), CPMpy, system prompt 3



Important gains with **sampling** in all LLMs (surprisingly?)

Evaluating Inference-Time Compute Methods

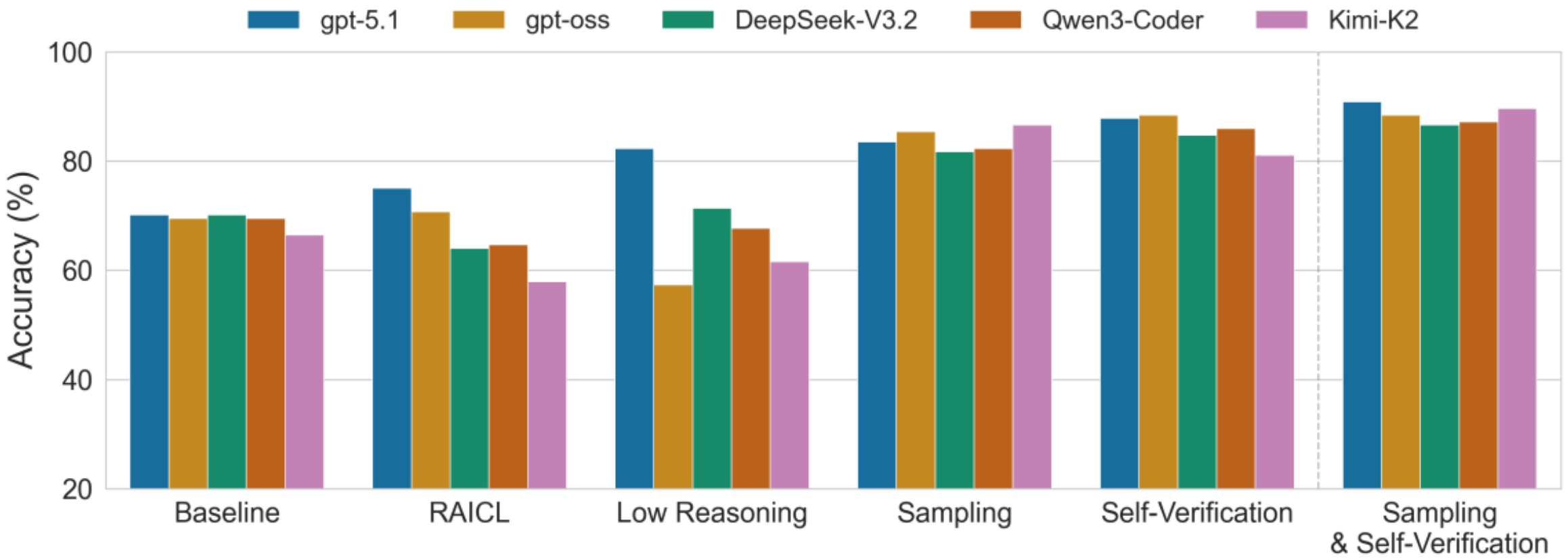
- Results on DCP-Bench-Open (164 instances), CPMpy, system prompt 3



Similar gains with **self-verification** of the generated model

Evaluating Inference-Time Compute Methods

- Results on DCP-Bench-Open (164 instances), CPMpy, system prompt 3



Best performing: **combine sampling with self-verification**!

Lesson learned #8: Towards agentic approaches

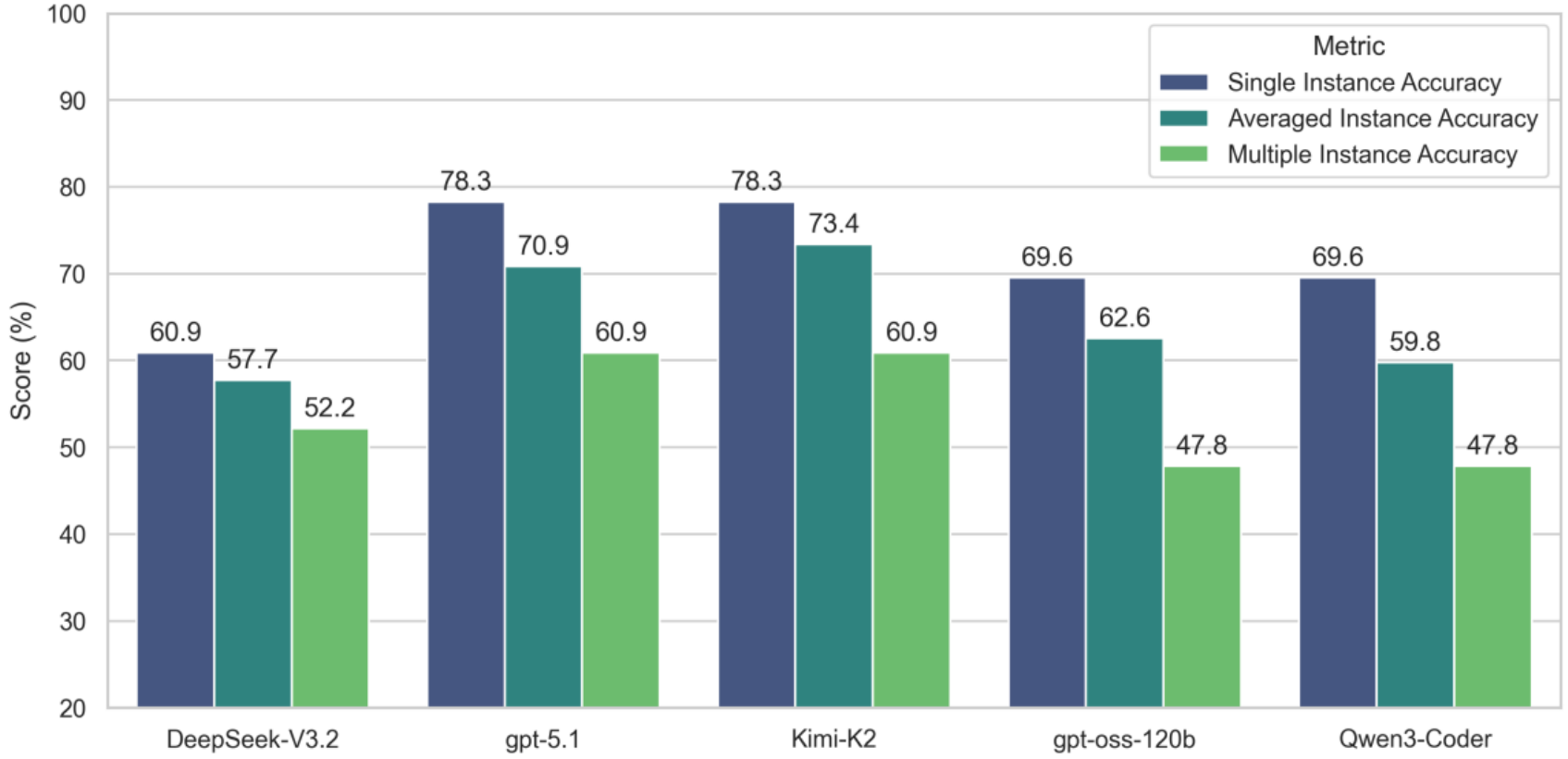
Are LLMs that good on modeling constraint problems?

Single-instance Solution Accuracy is not enough

Instances often change in real-world problems.

What about multiple data instances?

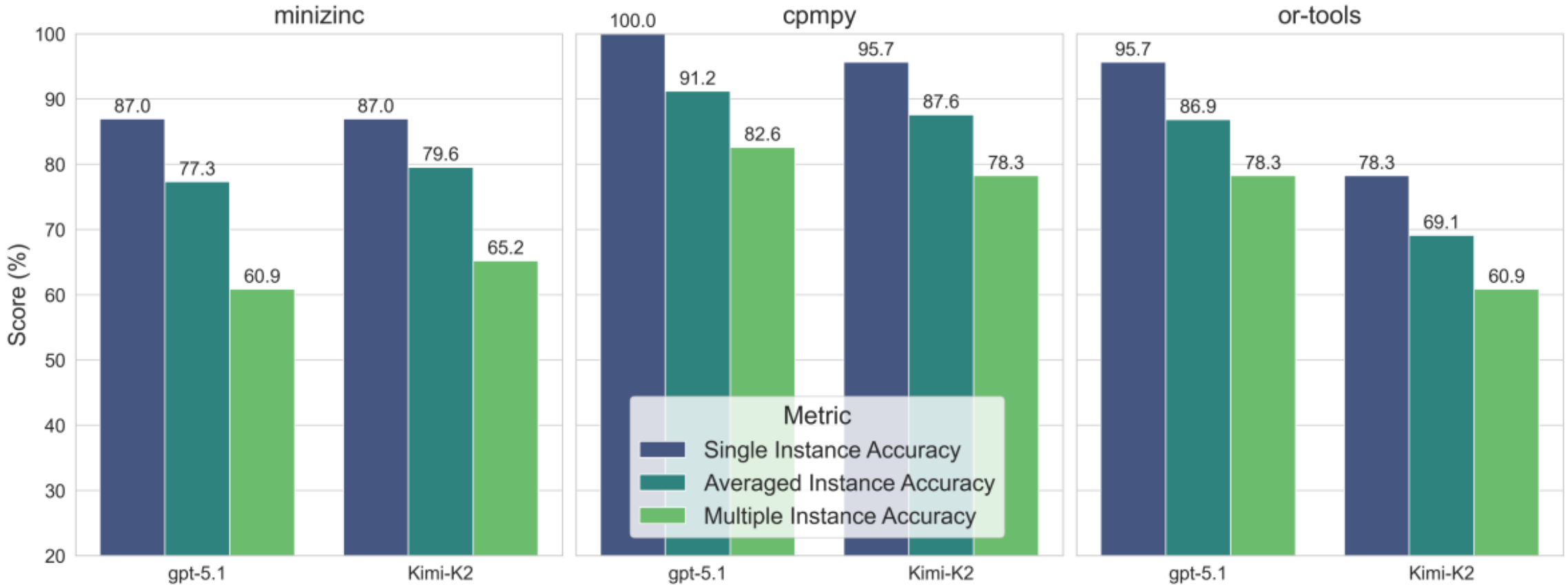
- Results on DCP-Bench-Open (23 problems), CPMpy, system prompt 3, baseline



Lesson learned #9: LLMs often overfit to the default data instance given in the prompt…

What about multiple data instances?

Results on DCP-Bench-Open (23 problems), sampl+selfver



LLMs often overfit to the default data instance given in the prompt… Even with more advanced techniques

Only a few datasets allow for multi-instance evaluation!

Most benchmarks still evaluate one fixed data instance per problem.

Bench4Opt	Multi — model-data and multi-parameter evaluation
CPEVAL	Multi — up to three instances per problem
CP-SynC	Multi — multiple instances with framework-agnostic evaluation
Text2Zinc	Multi enabled — separate model and data to allow additional instances
DCP-Bench-Open	Multi — multiple instances with framework-agnostic evaluation

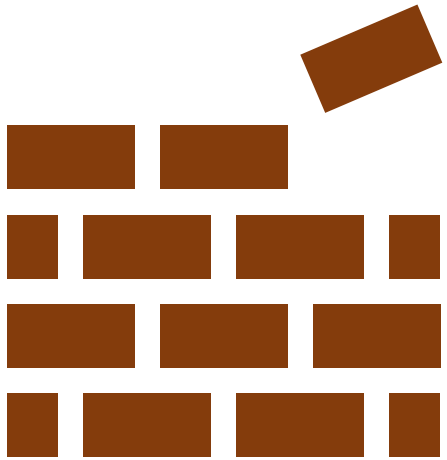
But results show that we should focus more on multi-instance evaluation to avoid overfitting (or getting “lucky” with the solution we got)

Lessons learned (so far): Summary

1. Evaluate solutions, not syntax: variable choices make model-to-model mapping fragile.
2. Use complex, diverse problems; homogeneous sets can overstate performance.
3. Datasets: Remove ambiguity from problem descriptions and solution-output schemas.
4. Datasets: Audit ground truth for hidden assumptions absent from the natural-language task.
5. Datasets: Remove extra constraints—such as symmetry breaking—that change feasible solutions.
6. LLMs perform better with Python-based, high-level modelling frameworks.
7. In-context learning is not reliably helpful when documentation is already strong.
8. Initial results point toward agentic workflows for modeling copilots.
9. Test multiple instances: LLMs often overfit the default data in the prompt.

DCP-Bench-Open

- Extending with more problems:
  - Currently: 164 *verified* & *diverse* instances
- **Multi-instance evaluation** and more metrics
- **Evaluating more frameworks** and modelling paradigms



DCP-Bench-Open is **OPEN** to contributions from the community:

✓ More problems
✓ More **instances** per problem
✓ More runners in **additional frameworks**
✓ Evaluation **metrics** (accuracy, tokens, efficiency …)

https://github.com/DCP-Bench/DCP-Bench-Open

Tutorial Agenda

Session A: Introduction, data, evaluation (9:00 - 10:30)

A1) Introduction (30 min): Tias Guns

A2) Benchmarking & Evaluation (60 min): Dimos Tsouros

Break (10:30 - 11:00)

Session B: Modelling copilots (11:00 - 12:30)

B1) Modelling copilots (60 min): Serdar Kadioglu

B2) Wrap-up and outlook (30 min): Tias Guns

Session B.1

Modeling Copilots

Presenter: Serdar Kadıoğlu

Definition: Modeling Co-Pilots

Let **I = (T, P, O, D)** represent the input specification where:

- **T:** Natural language problem description
- **P:** Set of input parameters with definitions, symbols, and shapes
- **O:** Set of output variables with specifications
- **D:** Metadata containing problem properties

Given **input I** and **data instance d**, learn function:

$$f : (I, d) \rightarrow M$$

where **M** represents the space of valid constraint models that correctly implement the specifications.

Complexity Spectrum: Difficulty depends heavily on the elements utilized for the mapping.

[Text2Model: Modeling Copilots for Text-to-Model Translation, S. Kadioglu et. al.]

Roadmap

Our roadmap covers **six research themes** from position papers vision to operational aspects. Each group builds on the last, forming a **coherent** agenda for next-generation Modeling Copilots.

Vision & Position Papers

Define the paradigm shift from solver-centric to human-centric modeling interfaces leveraging LLMs.

Foundations & Performance Baselines

Extract mathematical entities from text and establish diagnostic evaluation protocols.

Fine-Tuning & Synthetic Data Engines

Build verifiable data pipelines to adapt open-weights SLMs and fine-tuning for declarative modeling.

Structured Workflows & Multi-Agent Systems

Decompose complex specifications across specialized agent roles and persistent state.

Search, Verification & Ensembles

Frame formulation as a closed-loop search problem with exact solver feedback.

Human-in-the-Loop & Post-Solution Analytics

Support multi-turn preference elicitation, counterfactual querying, and explanation.

Part – I

Position & Vision Papers

Before diving into algorithms and details copilots, let’s visit the origins of the **vision, user needs, and value proposition** of Modeling Copilots.

Marks the paradigm shift from traditional **solver-centric approaches** to **human-centric modeling interfaces**
LLMs & conversational AI bridges domain expertise and mathematical formalism.

Seminal Position & Vision Papers

Position papers establish the **conceptual** and **empirical framing** for LLM-driven constraint modeling.
These span **system architecture, practitioner workflows, enterprise adoption**, and reframing of the classical vision.

Decision Optimization Copilot Manifesto

Wasserkrug et al. 2024

Outlines **system capabilities** for making optimization accessible to non-technical decision-makers via conversational AI: understanding, mental modeling, formal translation, tuning, and validation.

“Magical Box” Study

Lawless et al. 2025

Empirical **qualitative study** revealing that practitioners view optimization as opaque. Demonstrates that tools must foreground data handling, **iterative elicitation**, and stakeholder **trust** over pure algorithmic efficiency.

Democratizing Optimization

Simchi-Levi et al. 2025

Examines how GenAI transitions optimization from PhD specialist tools to **enterprise-wide** agentic systems. Accelerating prototyping from weeks to minutes while demanding formal mathematical guardrails.

Holy Grail 2.0

Tsouros et al. 2023

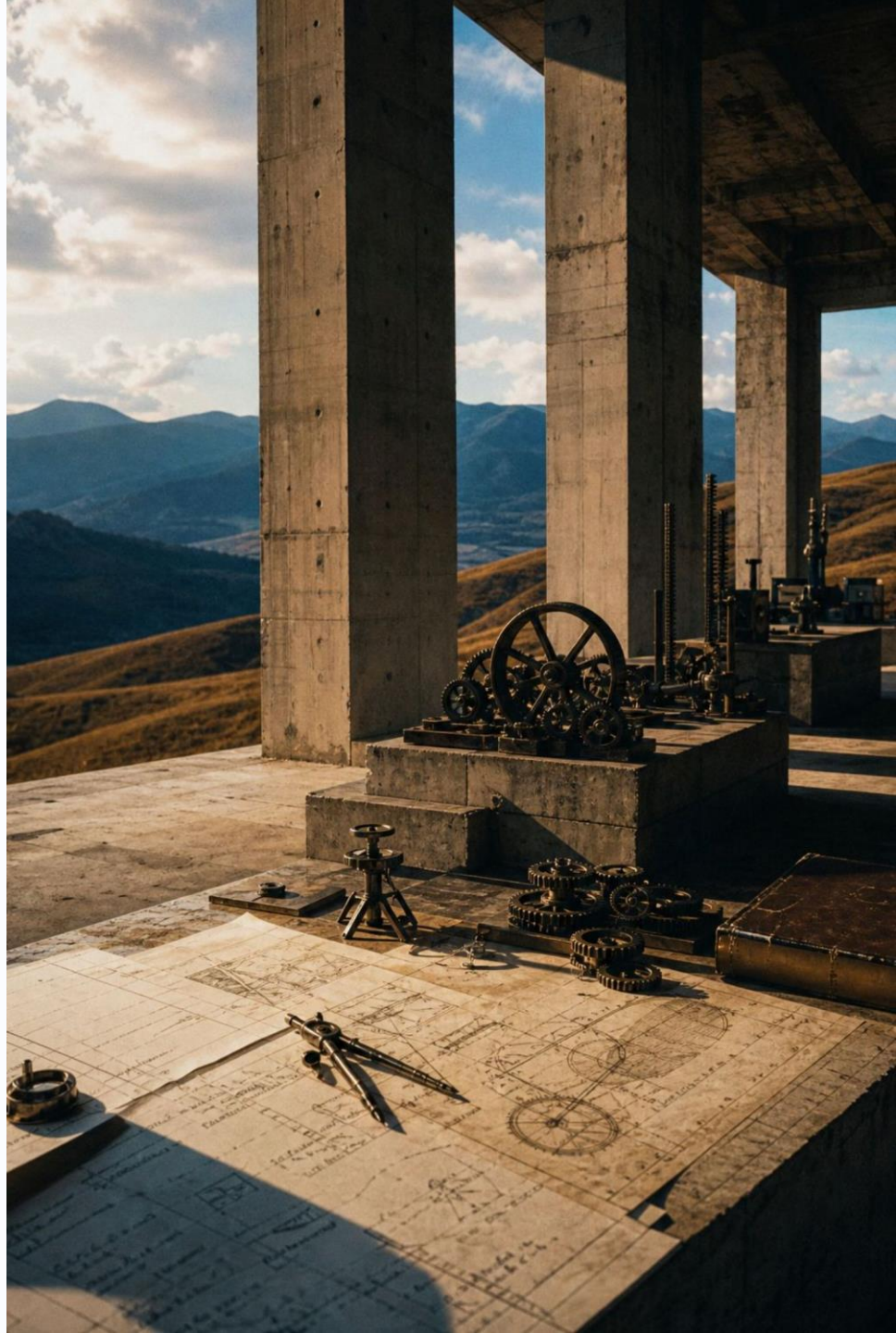
One of the earliest **blueprints**. Revisits Freuder's **classic CP vision** using modern LLMs. Proposes a 4-stage pipeline: Entity Extraction + Relation Graphing + Formal Specification + Executable Code + Debug Loop.

Part – II

Foundations & Performance Baselines

Following the overarching vision set, the next part is on the **technical layer**: extracting mathematical entities from text, defining **intermediate representations**, and establishing diagnostic benchmarks.

Robust copilots require principled parsing before any model synthesis can begin.



LaTeX2Solver: Hierarchical Document Parsing

*Ramamonjison et al. ACL'23 — **Focus: Hierarchical Document Parsing & Scientific Paper Auto-Formulation***

Core Challenge

Converting complex, **multi-page mathematical papers** and enterprise LaTeX specifications into executable solver code (Pyomo/Gurobi) — without losing structural context, symbol scope, or variable definitions buried across sections.

Hierarchical Parsing Architecture

1. **Document Tree Construction:** Extracts hierarchical relationships between sections, text blocks, and LaTeX math environments
2. **Entity & Symbol Alignment:** Links natural language parameter definitions directly to LaTeX mathematical symbols
3. **Code Synthesis:** Compiles parsed structures and variable domains into executable solver specifications

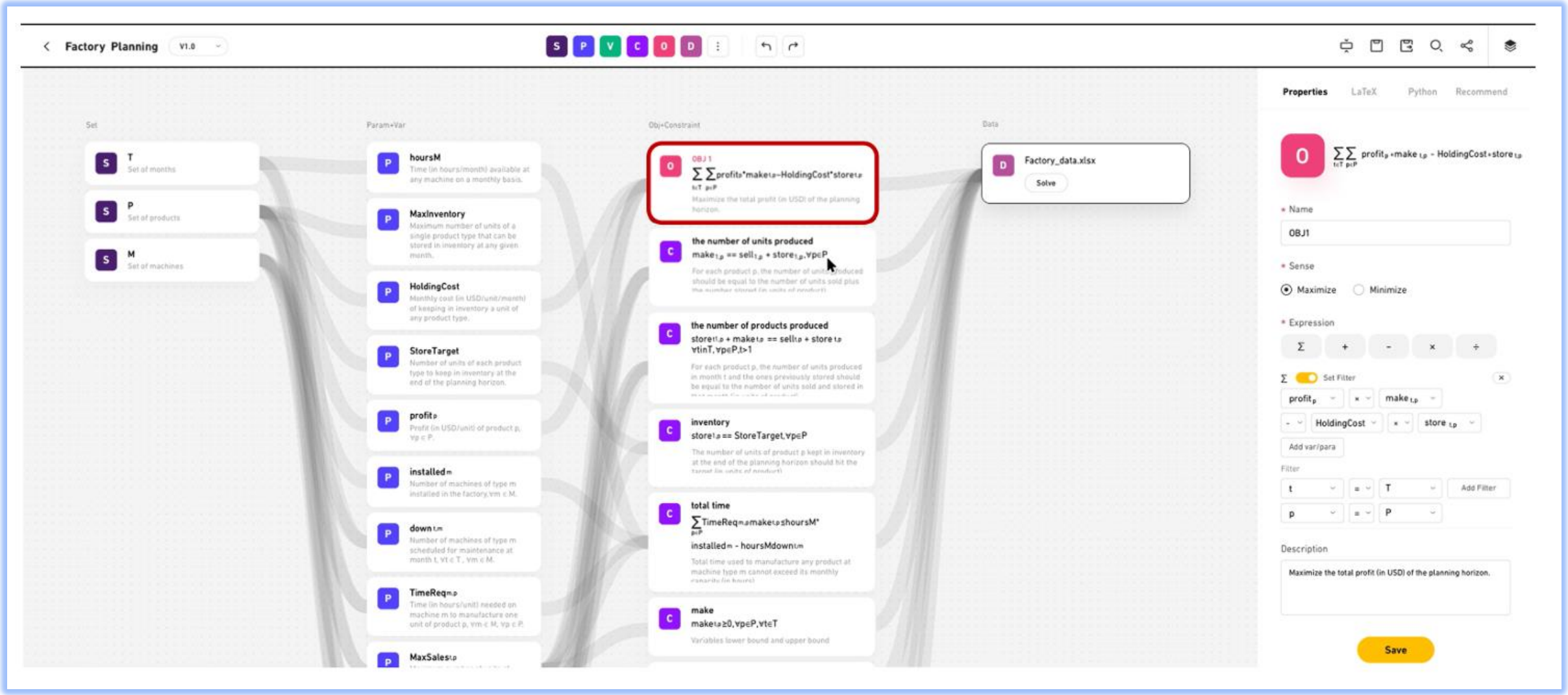
Insight

Treating LaTeX as flat text causes symbol ambiguity and variable scope loss. **Structural, hierarchical** parsing, separating section structure, prose context, inline math, and display equations is a **prerequisite** before feeding mathematical abstractions into LLM.

Takeaway

Scaling co-pilots to academic papers or enterprise specs **requires document structure parsing first**.

LaTeX2Solver: Hierarchical Document Parsing



Ner4Opt: Named Entity Recognition for Optimization

Kadıoğlu et al. CPAIOR'23 — ***Focus: Formal specification of NER for intermediate representations***

Core Idea

Introduces **NER specifically for mathematical optimization** automatically identifying and tagging core modeling building blocks from unstructured natural language text before any code synthesis begins.

Entity Schema

Decision Variables	Scalar Parameters
Objective Targets	Constraints & Bounds

Role in Copilot Pipelines

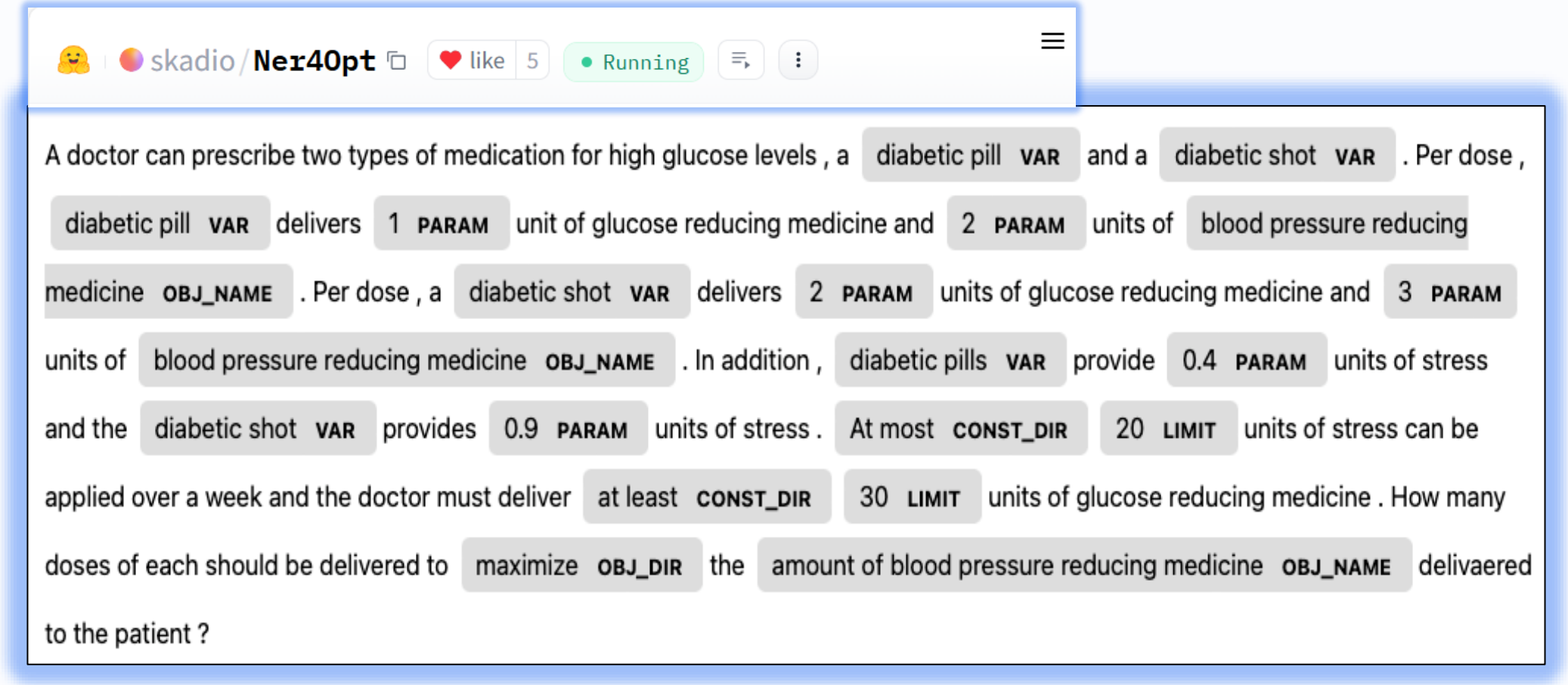
Acts as a foundational stage **Intermediate Representation.** Provide a deterministic entity schema upfront to prevent downstream LLMs from dropping parameters, conflating variable domains, or misinterpreting constraint scopes.

Takeaway

Extracting structured entities via **sequence tagging upfront** provides a deterministic schema that prevents dimensional and scoping errors during full model synthesis. This is a prerequisite to early LLMs. **50% Improvement in performance**

Ner4Opt: Named Entity Recognition for Optimization

Kadıoğlu et al. CPAIOR'23 — ***Focus: Formal specification of NER for intermediate representations***



CP-Bench: Evaluating LLMs for Constraint Modelling

Michailidis et. al. ECAI'2025 — ***Focus: Comprehensive Benchmarking for Constraint Programming***

A dedicated, systematic benchmark designed specifically to evaluate the **constraint programming modeling capabilities** of frontier LLMs across classic CP domains. Filling a critical gap in standardized evaluation in CP. Next iteration: **DSP-Bench-Open**

Compilation / Syntax Accuracy

Does the generated model compile without errors against solver grammar?

Feasibility / Correctness Accuracy

Does the solver find a solution that satisfies all stated constraints?

Execution Efficiency

Does the model solve within acceptable compute bounds across diverse CSPLib?

Key Diagnostic: **Significant performance drop-offs** occur when moving from simple linear models to **complex CP global** constraints **alldifferent, cumulative, circuit** and non-linear logic. These are the primary bottlenecks for frontier LLMs.

In-Context Learning for Constraint Modelling

*Michailidis, Tsouros, Guns, CP'24 — **Focus: In-Context Learning (ICL) & Prompt Engineering for CP***

Core Methodology

Investigates how systematically structured in-context learning prompts and **dynamic example selection** enable LLMs to synthesize executable constraint models without any fine-tuning. Evaluates standard LLMs on CP benchmarks using tailored prompt strategies incorporating:

- **Chain-of-Thought (CoT)** reasoning traces
- Variable domain definitions in prompt preamble
- Structural demonstration triplets (problem, formulation, code)

Critical Finding

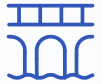
Few-shot demonstrations based on **structural similarity** matching variable/constraint hypergraphs drastically outperform surface-level textual similarity in retrieving relevant in-context examples. **Similarity at the network level**, not the word level, drives ICL success.

Takeaway

In-context learning combined with domain-aware demonstration design offers a **lightweight, training-free** mechanism for generating valid constraint models with no gradient updates required.

Text2Zinc: Cross-Domain Benchmarking in MiniZinc

*Kadıoğlu et. al. arXiv'25 — **Focus: Cross-Domain Benchmarking & Solver-Agnostic Evaluation***



Unified CP + OR Benchmark

Bridges **constraint satisfaction** and **mathematical optimization** in a single paradigm and solver-agnostic MiniZinc framework. The first unified cross-domain dataset of its kind.

Logic–Data Decoupling

Decouples abstract problem logic descriptions from concrete parameter data instances (.dzn), **preventing data memorization** during evaluation and ensuring true generalization testing.

Online Interactive Editor

Intentional effort on **data verification and curation** using an interactive editor with an AI assistant that has context into problem description and instance specification

Evaluation Metrics

Evaluates both **Execution Accuracy** (compiler validity) and **Solution Accuracy** (exact solution vector comparison) across paradigms and solvers and LLMs.

MiniZinc: Solver-Agnostic Modeling

High-Level Language

MiniZinc supports **both discrete and continuous optimization and satisfaction** problems with an intuitive, declarative syntax.

Multiple Backends

Paradigm and Solver-agnostic design communicates with CP, MIP, and SAT solvers through FlatZinc compilation—write once, run anywhere.

Global Constraints

Powerful abstractions like **all_different** simplify modeling, replacing numerous pairwise constraints with single declarations.

MiniZinc Example: All Different Constraint

```
% pairwise binary inequalities
all_different(array[int] of var int: x) =
    forall(i,j in index_set(x) where i < j)
        x[i] != x[j]

% built-in global constraint
all_different(array[int] of var int: x) =
    gecode_all_different(x) % native Gecode version
```

Global constraints allow users to **leverage higher-level abstractions** rather than focusing on low-level decomposition, significantly simplifying the modeling process.

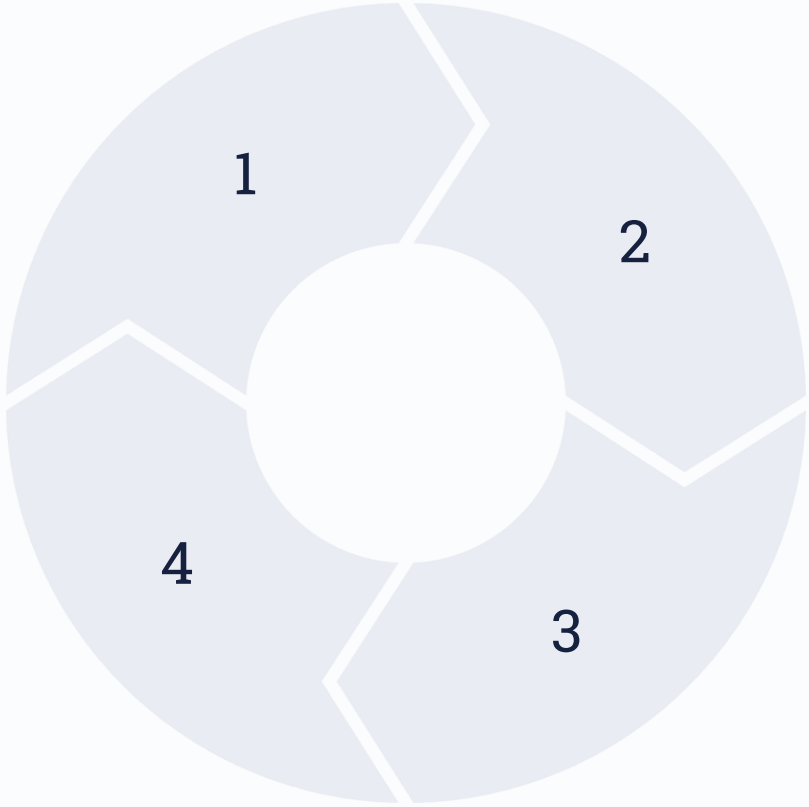
Text2Model: Copilot Taxonomy Simple to Complex

Kadıoğlu et. al. — ***Focus: Modeling Copilot Taxonomy & Intermediate Representations***

Provides a **systematic** design, evaluation, and taxonomy of LLM-based modeling strategies
Benchmarking the **full spectrum** from naive zero-shot generation to structured agentic workflows.

Out-of-the-box LLM

Vanilla prompting, zero-shot, few-shot performance. Single vs. Multi-Call



Structured Prediction

Grammar-based model generation to enforce LLM output

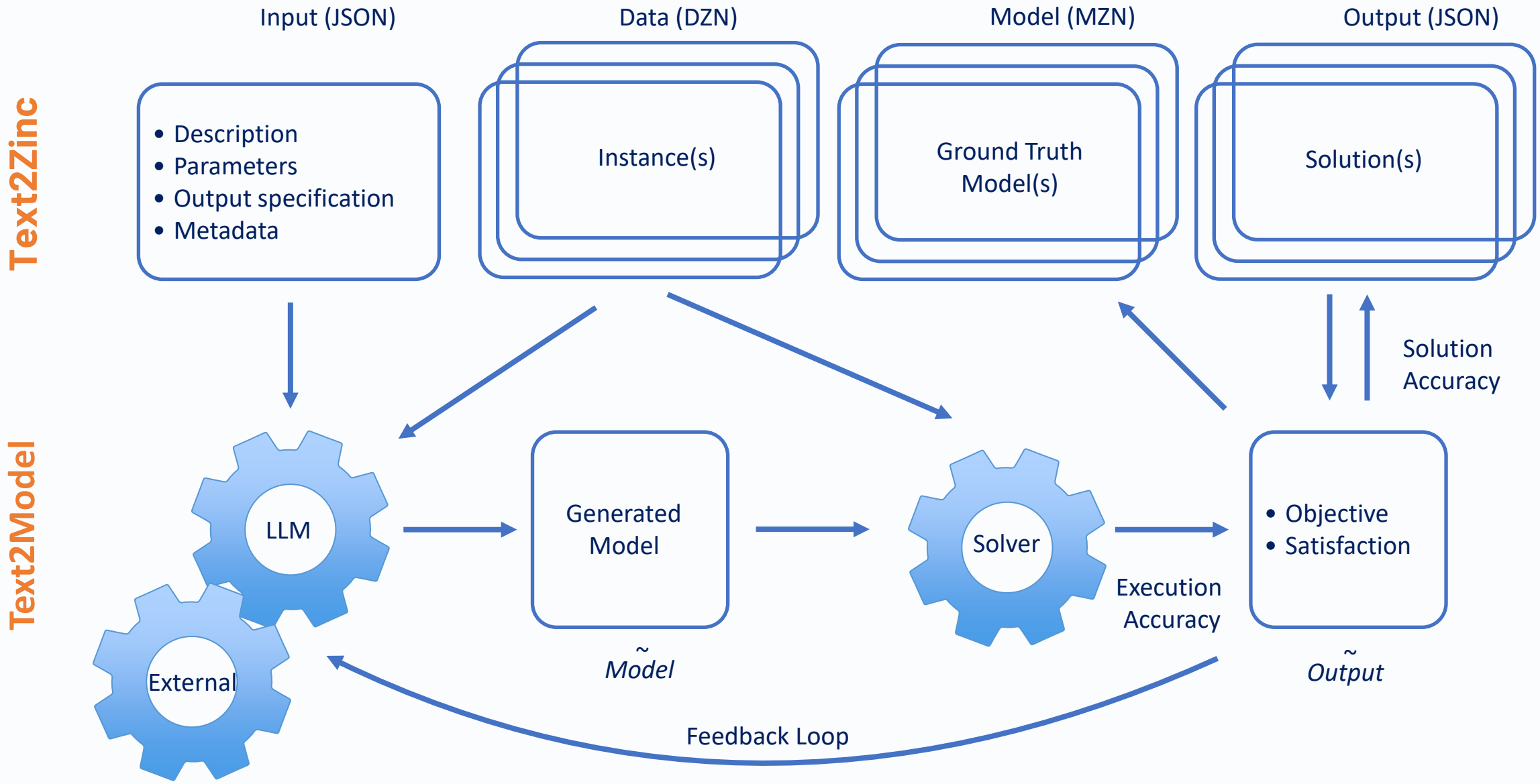
Knowledge Graph

Leveraging structured knowledge as intermediary representation

Chain-of-Thought & Agentic

Improved reasoning through step-by-step problem-solving and agentic decomposition

Text2Zinc Dataset & Text2Model Copilots



Knowledge Graph Generation

Given an optimization problem description and parameter nomenclature, the KG generation prompt instructs the LLM

1	**Identify Parameters** Extract each parameter with type, name, bounds, and alignment to predefined nomenclature.
2	**Identify Variables** Note types, names, and bounds; cross-verify with problem description for completeness.
3	**Identify Constraints** Detail each constraint with description, formula, associated variables, and classification (linear, nonlinear, global).
4	**Identify Objective** Define objective function, optimization type (min/max/satisfy), and relevant constraints.
5	**Generate TTL Representation** Construct **Turtle-format (TTL) KG** sequentially organizing parameters, variables, constraints, and objective.

Grammar Validation

When **syntax errors** are detected, the grammar validation prompt provides:

→ Problem Context

Problem description and data nomenclature for semantic grounding.

→ Current Code + Error

The syntactically incorrect MiniZinc code and **the specific error message from execution**.

→ MiniZinc Grammar Spec

The **formal grammar rules** from MiniZinc's Bison parser used to analyze and correct syntax.

Output: complete corrected MiniZinc code only, with all syntax errors resolved per the grammar specification.

MiniZinc Grammar Example

Grammar-constrained generation relies on formal grammar rules to validate generated code. Consider this **simple rule for Boolean literals:**

```
<bool-literal> ::= "false" | "true"
```

What This Rule Enforces

- Boolean values have exactly two valid forms
- No other spellings (False, TRUE, 0/1) are valid
- Prevents invalid expressions like `maybe` or `yes`

Why It Matters

Prevents model from generating invalid Boolean expr
Applied across the full grammar, this post-processing step catches a wide range of structural violations **without requiring access to token probabilities**.

Agentic Decomposition

Constraints

Generate constraint blocks ensuring global correctness.

Objective

Generate the solve statement for optimization.

Parameters & Variables

Declare all inputs and decision variables.

Stitch

Combine sections into a complete MiniZinc model.

Each sub-task has dedicated principles: parameters/variables focus on type consistency and bounded declarations; constraints emphasize global constraints and iteration correctness; objective ensures a single solve keyword; **stitch validates integration, resolves circular dependencies**, and confirms type consistency across all sections.

Benchmark Results: GPT-5.2

Strategy	NLP4LP E/S	CSPLIB E/S	HAKANK E/S	INDUSTRYOR E/S	MAMO-EASY E/S	MAMO-COMP E/S
Zero-Shot	63.08 / 32.31	63.64 / 36.36	50.00 / 31.82	66.00 / 46.00	95.71 / 79.75	48.82 / 27.96
Chain-of-Thought	70.77 / 35.38	54.55 / 27.27	59.09 / 36.36	80.00 / 49.00	98.62 / 83.90	57.35 / 39.34
CoT + Grammar	**76.92 / 37.26**	**90.91 / 45.23**	**72.73 / 37.9**	**92.00 / 56.00**	**99.85 / 83.74**	**95.26 / 54.03**
Agentic + Code	76.92 / 29.23	81.82 / 36.36	68.18 / 36.36	86.00 / **57.00**	99.23 / **84.97**	92.89 / 54.03

CoT + Grammar achieves the best results across most datasets, potential of structured output generation.

Frontier Model + CoT + Grammar Encoding leads but..

1271	956	362
CoT + Grammar Exec acc	CoT + Grammar Sol acc	Unsolved Instances
Best execution accuracy	Best solution accuracy	27% remain beyond current LLM capabilities

27% of instances remain out of reach

Online Leaderboard: https://huggingface.co/spaces/skadio/text2model-leaderboard

AGENTS & AUTOMATED ASSISTANTS

LLM Copilots for Text-to-Model Translation

A suite of LLM modeling copilots, datasets, fined-tuned models, demos, interactive editor, and online leaderboard for translating natural language text into formal combinatorial constraint models.

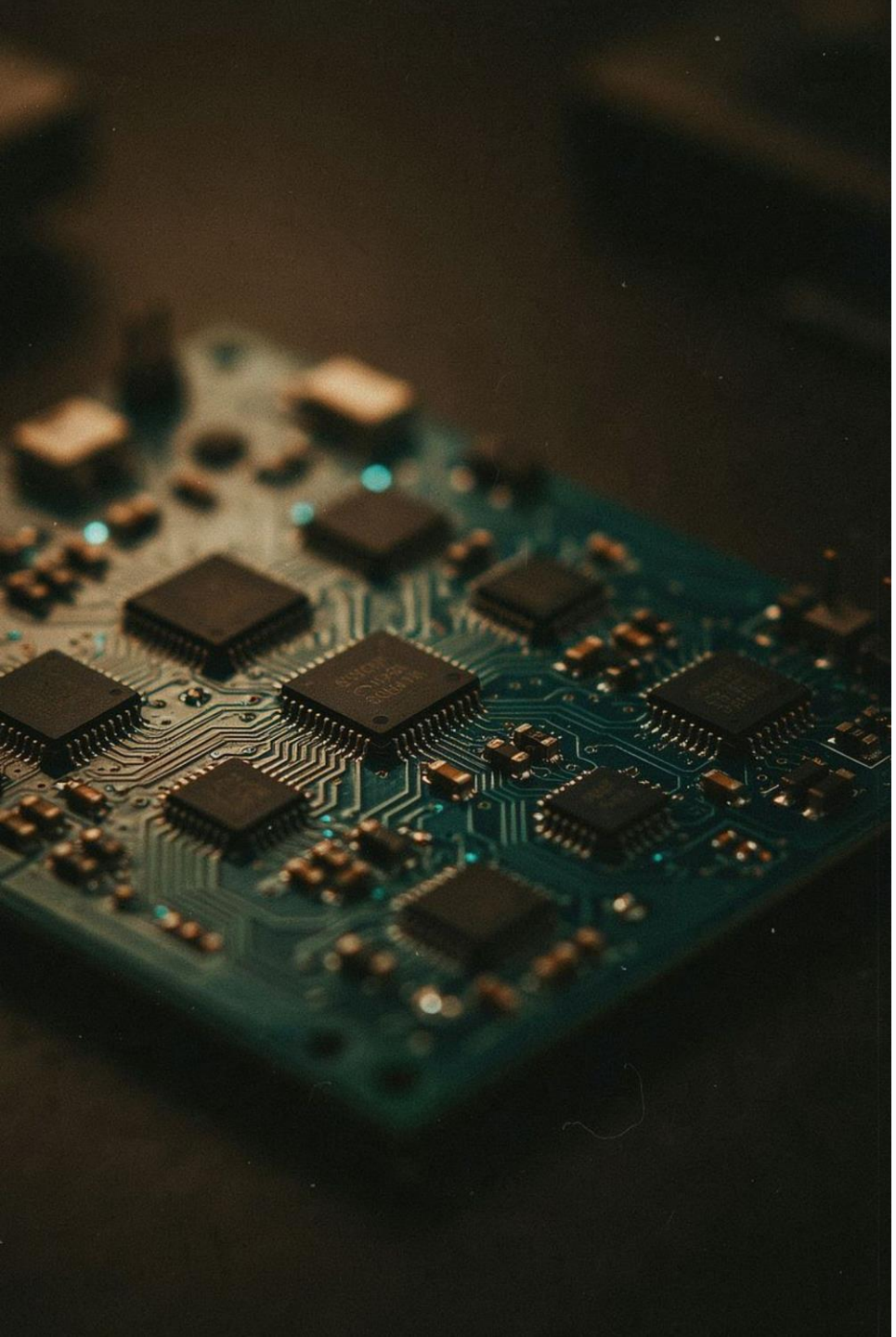
Serdar Kadıoğlu[1,2] Karthik Uppuluri[2]

[1] Department of Computer Science, Brown University [2] AI Center of Excellence, Fidelity Investments

Paper | Copilots | Slides | Leaderboard | Interactive Editor | BibTeX

```
INSTALL pip install text2model
```

https://skadio.github.io/text2model



Part – III

Fine-Tuning & Synthetic Data Engines

Base LLMs have virtually **zero native exposure** to declarative modeling languages like MiniZinc.

Address the data bottleneck head-on building **verifiable synthetic data engines** and **compiler-driven error bootstrapping pipelines** to adapt open-weights small language models (SLMs) to the constraint modeling.

ORLM: A Customizable Framework for OR Fine-Tuning

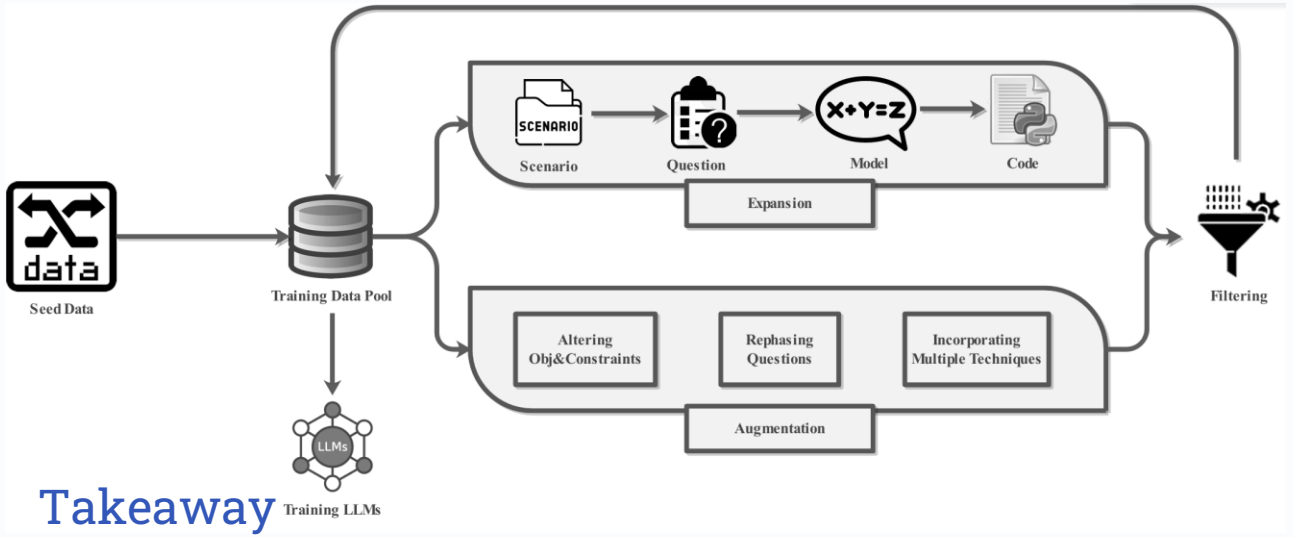
Huang, Tang, Hu, Jiang, Zheng, Ge, Wang, Wang , 2024 — ***Focus: Fine-Tuning & Synthetic Data Engines***

Core Idea

An open-source, fully customizable framework for training foundation models (Llama, Qwen) specifically on OR tasks **OR-Instruct dataset**, achieving performance that consistently surpasses general-purpose frontier LLMs on OR benchmarks.

Synthetic Data Pipeline

- **Problem mutation:** Systematically varies parameters, domains, and constraint types from seed problems
- **Back-translation:** Generates natural language from verified solver code to create paired training examples
- **CoT instruction tuning:** Separates parameter parsing, variable declaration, constraint formulation, and code compilation into explicit reasoning steps



Takeaway

Domain-specific fine-tuning on specialized mathematical schemas **drastically reduces COPT API call syntax errors** and variable dimension mismatches

Industry-OR as a challenging OR tasks dataset.

Verifiable Synthetic Data for Trustworthy Agents

Lima, Phan, Kalagnanam, Patel, Zhou — ***Focus: Verifiable Data Generation & Agent Reliability***

The Core Bottleneck

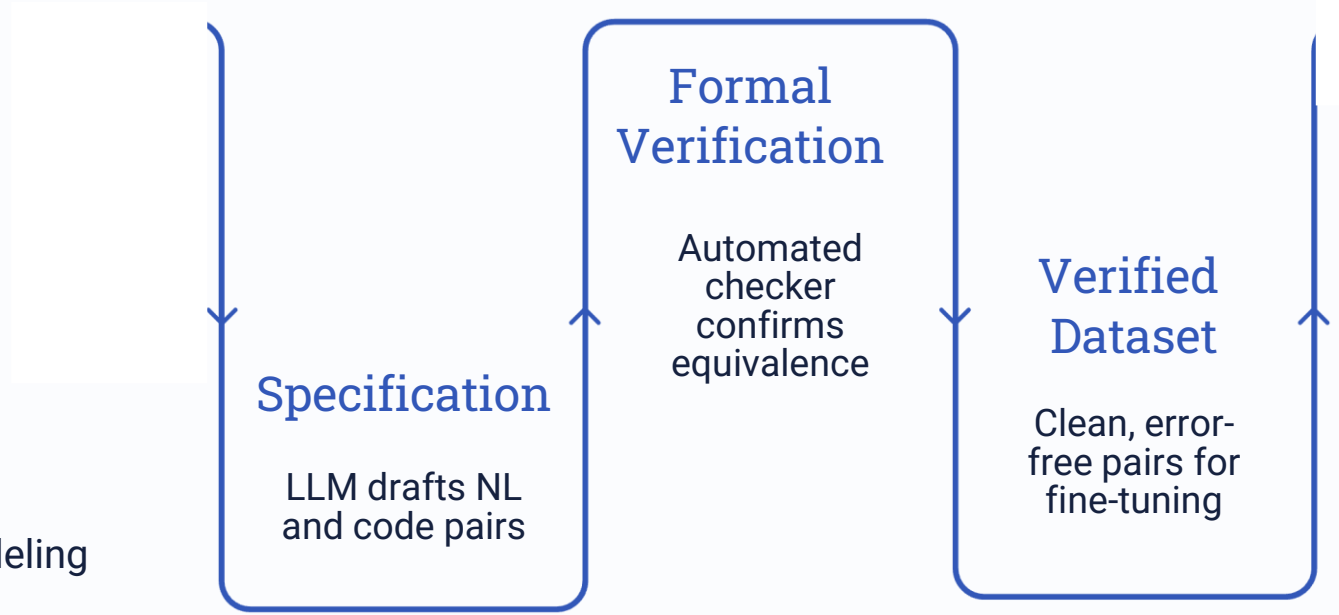
Unreliable synthetic training data is a critical bottleneck for optimization agents.
Noisy, mathematically incorrect <natural language, code pairs> degrade model trustworthiness during fine-tuning.

Verifiable Pipeline Mechanism

Starting the structured symbolic representation to guarantee that every synthetic NL-to-model pair accurately matches its executable solver output before being added to the training corpus.

Downstream Impact

OptiTrust eliminating erroneous pairs directly improves execution success rates. High-performing, trustworthy modeling agents depend on **verifiable, error-free synthetic training datasets**, not just large data.



MiniZinc is Out-of-Distribution for SLMs

Complete Failure Without Fine-Tuning

The combination of **rare syntax** and **unfamiliar paradigm** explains the near-zero performance of untuned models.

0%

Qwen3, LLaMa3, Gemma2, GPT-OSS

Execution accuracy out-of-the-box

First Systematic Study

Fine-tuning SLMs for MiniZinc code generation + Blueprint

Learn2Zinc: Fine-Tuning SLMs for MiniZinc

Kadıoğlu et al., 2026 — ***Focus: Domain Adaptation & Small Language Model (SLM) Fine-Tuning***

The Syntax Barrier Problem

MiniZinc is a rare, domain-specific language with minimal representation in LLM pretraining corpora. General-purpose SLMS produce **near-zero execution accuracy** out of the box.

Core Investigation

Can fine-tuning small open-weights language models (0.6B–20B parameters) overcome the syntax barrier in rare, domain-specific languages like MiniZinc, where **pre-training data is virtually absent?**

Cross-Model Error Bootstrapping

Constructs specialized training sets by collecting real **syntax error traces** from multiple LLMs, explicitly teaching SLMs how to recognize and self-correct compiler errors through negative examples

98%

Execution Accuracy

Achieved when ensembling fine-tuned SLMs — up from near-zero baseline

35%

Solution Accuracy Cap

Semantic / combinatorial reasoning ceiling — revealing a distinct challenge beyond syntax

The Core Insight: Verification Enables Fine-Tuning

Text2Model copilots on hundreds of **Text2Zinc** instances generates a **large volume of MiniZinc models** even if not manually written.

The critical advantage is the **verification loop**:

- Optimization problems: verified against known objective values
- Satisfaction problems: feasibility asserted

This yields a large pool of **verified (text, model) pairs** that makes fine-tuning possible *without requiring manual annotation*.

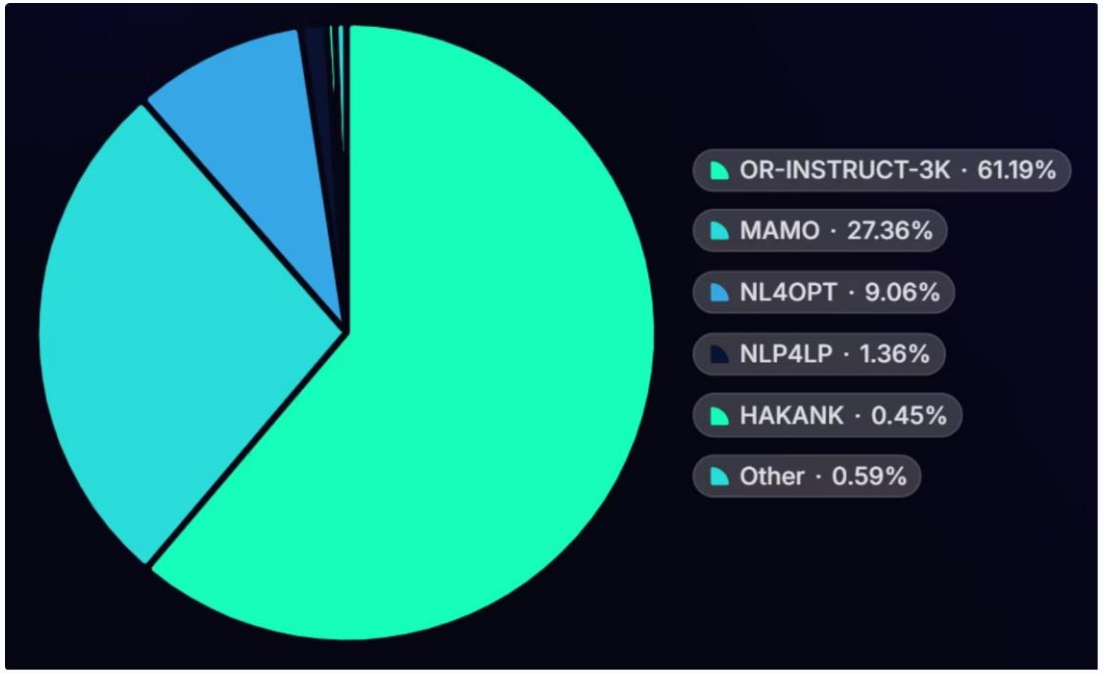
Building the Fine-Tuning Dataset

Our fine-tuning dataset combines verified **MiniZinc solutions from Text2Zinc** with translations from the **OR-INSTRUCT dataset** (Huang et al. 2024).

1	2	3
OR-INSTRUCT With COPT Models in Python	Translate via GPT-5.2	Verify & Add

OR-INSTRUCT contains optimization problems with **COPT models** in Python.
GPT-5.2 translates these to MiniZinc; only verified translations are added to the dataset.

Fine-Tuning Dataset: 2,208 Problems with 8,014 Pairs



Summary

2,208 unique problems across **7** sources.

Instances may have **multiple alternative MiniZinc models** from different Text2Model copilot strategies.

8,014 instruction-tuning (text, model) pairs for fine-tuning small language models.

Fine-Tuning Methodology

Our methodology combines **multiple SLM families** at varying parameter sizes with **two fine-tuning objectives**: generation fine-tuning and error-correction fine-tuning.

1) Generation Fine-Tuning

Teach SLMs to produce valid MiniZinc from natural language descriptions.

2) Error-Correction Fine-Tuning

Teach SLMs to identify and **fix syntax mistakes** in broken MiniZinc code.

Small Language Models

Qwen3-0.6B

Smallest model tests **lower bound** of SLM capability.

LLaMA-3.2-1B & 3B

Mid-range models from Meta's LLaMA family.

Gemma-2-9B

Google's 9B parameter model.

GPT-OSS-20B

Largest model tests **upper bound** of SLM capability.

Three Fine-Tuning Datasets

Learn2Zinc-Base

8,014 pairs of (text, model)

Direct **code generation** training.

Learn2Zinc-CoT

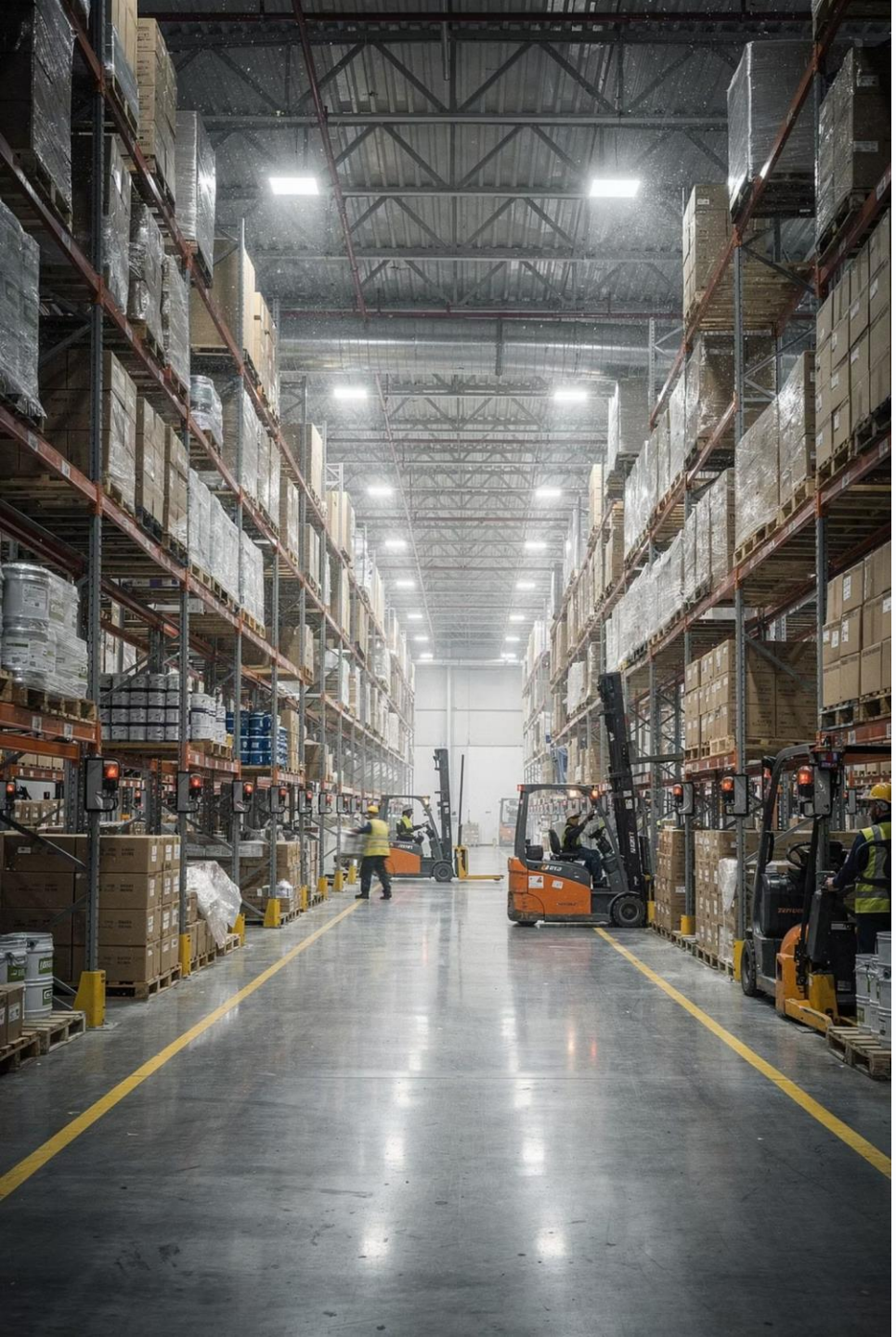
8,014 pairs of (text, reasoning, model)

GPT4o generates **reasoning chains** identifying variables, parameters, constraints, and objectives.

Learn2Zinc-Base+CoT

16,028 pairs combining both Base and CoT

Mixed training strategy.



Benchmark: Text2Zinc-IndustryOR

100 Text2Zinc-IndustryOR instances as test set
the most challenging benchmark as reported across multiple studies.

86%

Best Exec Accuracy

CoT + Grammar GPT-5.2 (Kadıoğlu et al. 2026)

57%

Best Sol Accuracy

CoT + Grammar GPT-5.2 (Kadıoğlu et al. 2026)

NOT included in fine-tuning

Model	Strategy	Exec Acc (%)	Sol Acc (%)
Qwen3-0.6B	Original-8bit	**0.0**	**0.0**
	Learn2Zinc-Base	**51.0**	9.0
	Learn2Zinc-CoT	44.0	10.0
	Learn2Zinc-Base+CoT	51.0	12.0
LLaMA-3.2-1B	Original-8bit	**0.0**	**0.0**
	Learn2Zinc-Base	**44.0**	4.0
	Learn2Zinc-CoT	22.0	1.0
	Learn2Zinc-Base+CoT	46.0	4.0
LLaMA-3.2-3B	Original-8bit	**0.0**	**0.0**
	Learn2Zinc-Base	**62.0**	10.0
	Learn2Zinc-CoT	47.0	9.0
	Learn2Zinc-Base+CoT	58.0	12.0
Gemma-2-9B	Original-8bit	**0.0**	**0.0**
	Learn2Zinc-Base	**74.0**	**22.0**
	Learn2Zinc-CoT	49.0	15.0
	Learn2Zinc-Base+CoT	70.0	21.0
GPT-OSS-20B	Original-4bit	6.0	5.0
	Learn2Zinc-Base	**66.0**	**27.0**
	Learn2Zinc-CoT	52.0	21.0
	Learn2Zinc-Base+CoT	63.0	27.0

Near-Zero Baseline

Remarkable Boost | COT underperforms

SLMs vs. Frontier (86%-57%)

TEACHING SYNTAX

The Syntax Bottleneck

Initial fine-tuning reveals a consistent pattern: **SLM outputs are riddled with syntax errors**. Even the best execution accuracy is only 76% before we can evaluate semantic correctness, most outputs fail to compile.

This shifts our focus: **how can we teach SLMs to avoid and correct their syntax mistakes?** We move from generation fine-tuning to error-correction fine-tuning.

Shifting from Generation Fine-Tuning

Phase 1: Generation

Teach model to produce MiniZinc

Phase 1 FT

Fine-tune on correct examples

Phase 2: Error-Correction

Train on common syntax mistakes

Phase 2 FT

Fine-tune to recognize and fix errors

Design an augmented training strategy that **explicitly teaches syntax correction**. The key insight: models need exposure to common errors *and* ***their fixes***, not just correct outputs. This requires a dedicated error-correction dataset built from **two complementary sources.**

Error Correction Dataset: Synthetic Corruptions

We derive **20 corruption rules** from MiniZinc's BNF grammar, organized into three difficulty levels, to generate realistic syntax errors and their fixes.

Easy (7 rules)

Delimiters and punctuation: missing semicolons, dropped commas, unbalanced brackets.

Medium (6 rules)

Keywords and types: invalid solver directives, dropped `var` or `of` keywords, split compound tokens like `endif`.

Hard (7 rules)

Structural elements: swapped declaration order, invalid operators, misplaced semicolons.

Sampling: 30% easy, 30% medium, 30% hard, 10% identity (no fix needed).

This process yields **4,452 verified instances** with **⟨text, corrupt_model, model⟩** triples.

Taxonomy of Corruption Rules

Rule ID	Difficulty	Description	Example
E1_drop_semicolon	Easy	Remove trailing semicolon	`x = 5; → x = 5`
E2_drop_comma	Easy	Remove comma from list	`[1, 2, 3] → [1 2, 3]`
E6_drop_close_paren	Easy	Remove closing parenthesis	`sum(x) → sum(x`
M1_drop_var	Medium	Remove "var" keyword	`var int: x → int: x`
M2_drop_of	Medium	Remove "of" keyword	`set of int → set int`
M3_solve_keyword	Medium	Replace maximize/minimize	`maximize → max`
M4_capitalize_keyword	Medium	Capitalize MiniZinc keyword	`constraint → Constraint`
H1_drop_then	Hard	Remove "then" from if-then-else	`if x > 0 then y → if x > 0 y`
H3_swap_type_ident	Hard	Swap type and identifier	`var int: x → x: var int`
H5_wrong_logic_op	Hard	Replace /\ with && (C-style)	`x /\ y → x && y`

Cross-Model Error Bootstrapping

Synthetic corruptions may not capture actual LLM failure modes. We address this by collecting real errors from model runs:

01

Run All 15 SLMs

Across all 5 SLMS and 3 fine-tuning strategies with **sampling temperatures** (0.2, 0.5, 0.8) to maximize error variety.

02

Collect Execution Failures

For each **⟨text, corrupt_model, error_message⟩**, GPT-5.2 to make **minimal targeted fixes** preserving original intent.

03

Verify & Add to Dataset

If the fix compiles and produces correct output: add **⟨text, corrupt_model, model⟩**. Yields **2,286 verified correction instances**.

04

Collect Successes Too

Execution and solution successes yield new positive (text, correct_model) pairs — extending the base dataset to **8,911 generation instances**.

Augmented Dataset with Dual Fine-Tuning Objective

4,452

Synthetic Corruptions

Grammar-based error-correction pairs

2,286

Cross-Model Corrections

Real LLM failure corrections via GPT-5.2

8,911

Generation Examples

Verified (text, model) pairs for code generation

15,649

Total Instances

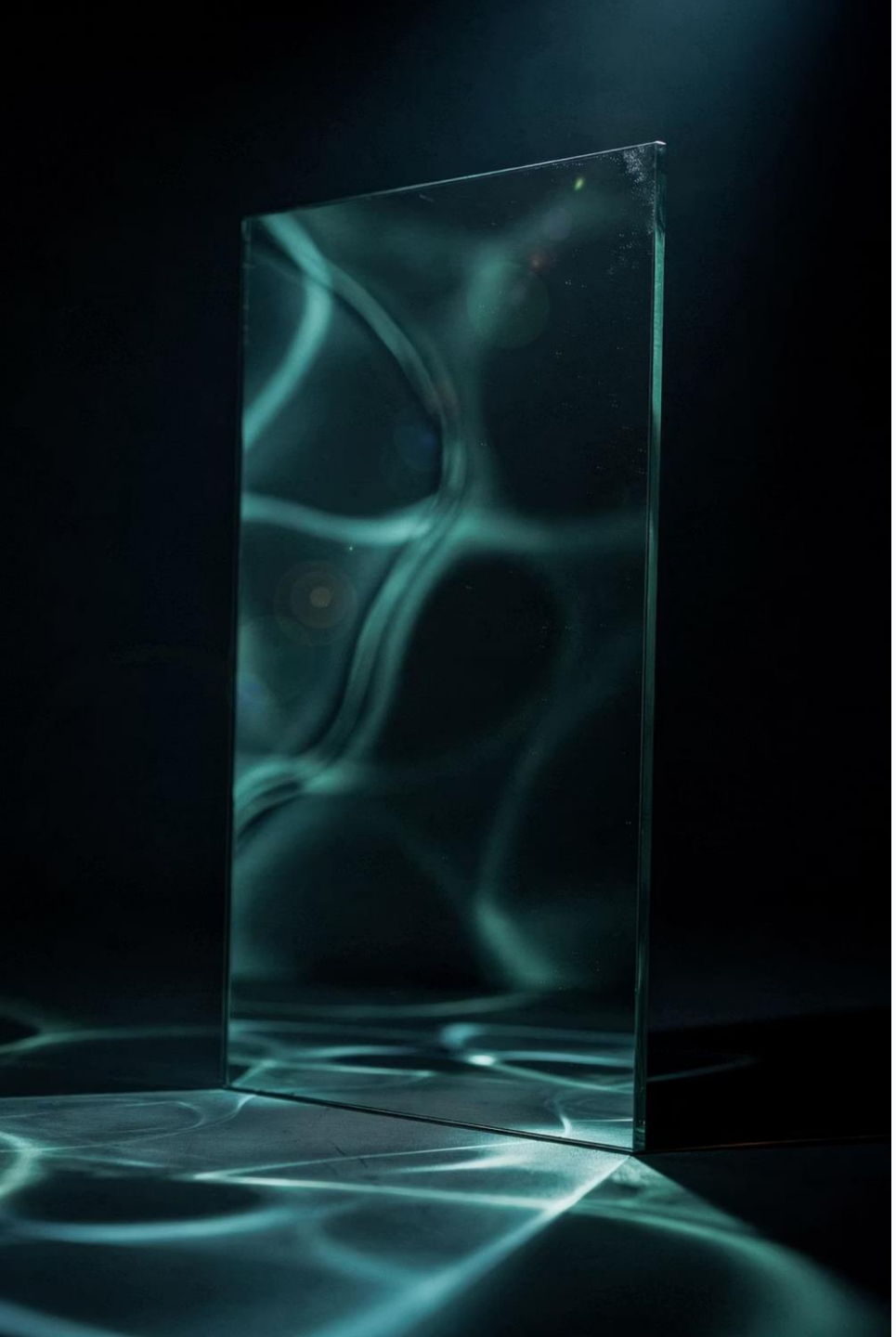
Combined augmented fine-tuning dataset

Learn2Zinc Augmented

Fine-tuning all five SLMs on the 15,649-instance augmented dataset with **dual objectives** (generation + error-correction) yields **consistent improvements** across all model sizes.

Model	Orig Exec (%)	Base Exec (%)	Aug Exec (%)	Aug Sol (%)
Qwen3-0.6B	0.0	51.0	**64.0**	13.0
LLaMA-1B	0.0	44.0	**57.0**	8.0
LLaMA-3B	0.0	62.0	**70.0**	17.0
Gemma-9B	0.0	74.0	72.0	22.0
GPT-OSS-20B	6.0	66.0	**76.0**	**32.0**

Validating that training for error correction directly addresses the syntax bottleneck.



Beyond Zero-Shot: Self-Reflection & Ensemble

Having fine-tuned for error correction, we now leverage this capability at **inference time**. Two complementary strategies

Self-Reflection Loop

Model receives its broken code + compiler error message and retries — up to 5 attempts.

Top-Down Ensemble

Models tried in descending capability order: GPT-OSS-20B → Gemma-9B → LLaMA-3B → LLaMA-1B → Qwen-0.6B.

A Key Milestone: On Par with GPT-5.2 (86%)

89%

GPT-OSS-20B augmented + self-reflection execution accuracy

Our Augmented SLM

GPT-OSS-20B: **6%** → **76%** → **89%** with self-reflection. Achieves on-par execution accuracy with its frontier variant.

Frontier GPT-5.2

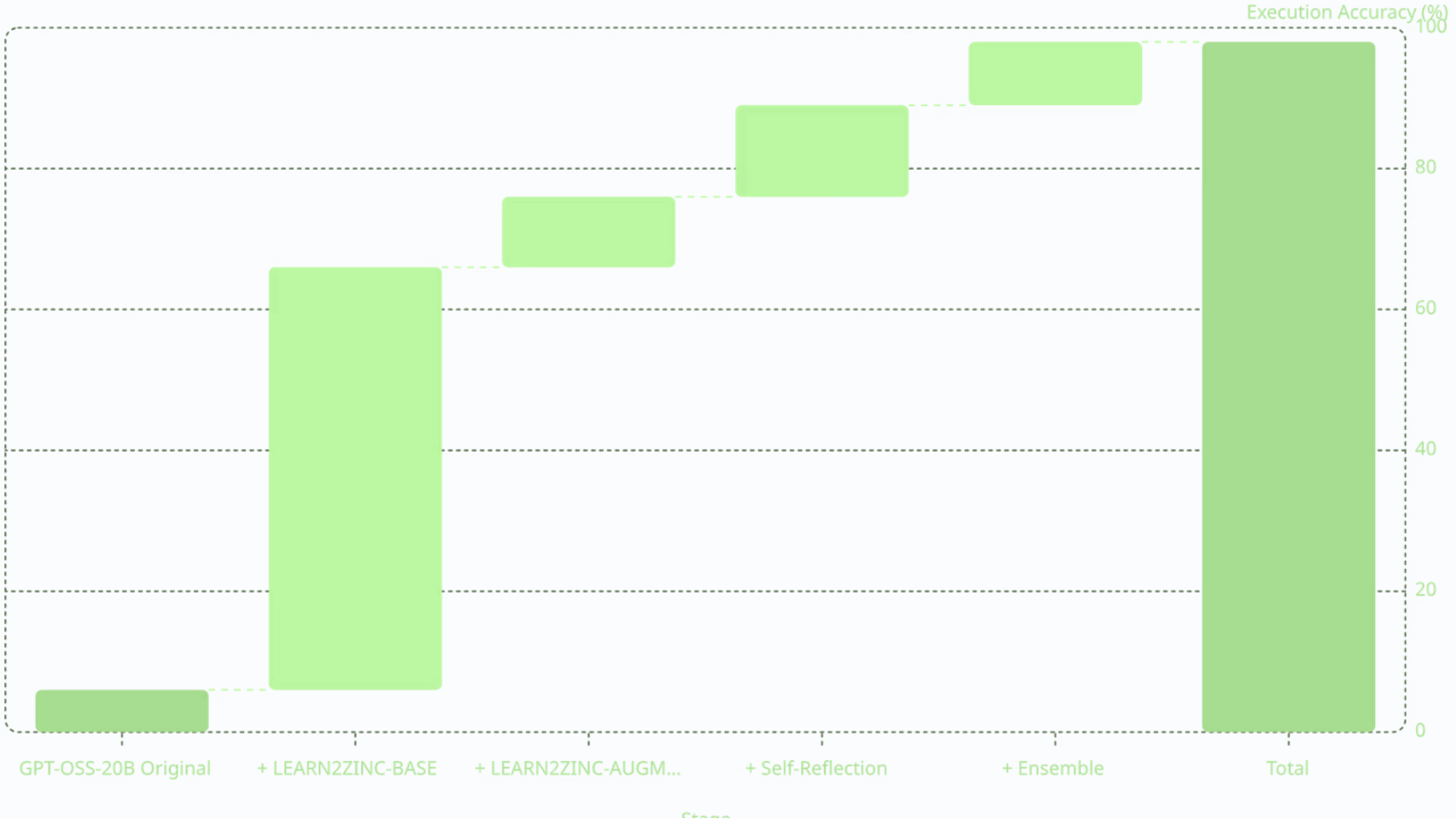
86% execution accuracy with the best Agentic copilot. Our fine-tuned SLM **matches and exceeds** this benchmark.

Syntax errors are fixable through retry. Semantic errors are not. Solution accuracy remains flat with retries.

98% Execution Accuracy with Ensemble

Model	Approach	Exec (%)	Sol (%)
GPT-OSS-20B	Original-4bit	6.0	5.0
GPT-OSS-20B	Learn2Zinc-Base	66.0	27.0
GPT-OSS-20B	Learn2Zinc-Augmented	76.0	32.0
GPT-OSS-20B	Aug + Self-Reflection	89.0	34.0
Our Fine-Tuned SLMs	**Learn2Zinc – Ensemble**	**98.0**	35.0
GPT-5.2	CoT + Grammar (Kadioglu et. al.)	86.0	**57.0**

Learn2Zinc: Overall Progression



Each stage of Learn2Zinc **builds on the previous, steadily improving** execution accuracy from **6%** to a remarkable **98%** effectively **solving the MiniZinc syntax problem** for small language models.

Learn2Zinc: Key Take Aways

Novelty

Cross-model error bootstrapping constructs realistic syntax error-correction datasets from LLM failures, enabling models to learn both code generation and error correction with **dual fine-tuning**

Result

Learn2Zinc-Augmented + Reflection Ensemble achieves **98% execution accuracy** effectively resolving the syntax bottleneck. Solution accuracy saturates at 34–35%.

Insight

Unlike LLMs, **COT fine-tuning does not improve performance** when foundational syntax knowledge is absent in SLMs. Syntax can be learned; reasoning remains the primary challenge.

→ Distillation Reasoning from Larger Models

Distilling **constraint reasoning capabilities** from frontier models into smaller, more efficient SLMs.

OptiMind: Teaching LLMs to Think Like OR Experts

*Zhang, Chen, Zope, Barbalho, Mellou, Molinaro, Kulkarni, Menache, Li — **Focus: Domain Expert Cognition & Pattern Modeling***

Rather than relying on unstructured generation, OptiMind embeds **human OR expert domain knowledge** across its four-stage framework: combining class-based error profiling, data cleaning, fine-tuning, and error-aware inference.

Class-based Error Profiling

OptiMind categorizes optimization problems into 53 canonical classes (e.g., TSP, set cover) to identify structural failure patterns and generate targeted domain hints.

Automated Data Cleaning

Combines LLM reasoning, class-specific error hints, and majority voting to **repair missing parameters** and filter invalid benchmark instances.

Domain Supervised Fine-Tuning

Fine-tunes a 20B open-source model (GPT-OSS-20B) on the cleaned, domain-verified optimization dataset.

Error-Aware Test-Time Inference

Uses class-specific hints alongside optional self-consistency and solver-execution feedback for **dynamic self-correction** during model inference.

Learn2Zinc & OptiMind

OptiMind (Zhang et al. 2026)

Fine-tunes **GPT-OSS-20B** for optimization modeling in Python with Gurobi Solver.

Kadioğlu et al. (2026) finds that OptiMind fine-tuned GPT-OSS-20B has its **MiniZinc capabilities reduced to zero**, unintended consequence of fine-tuning.

Attention needed in Fine-Tuning

Fine-tuning models for one domain-specific language may **degrade capabilities in others**.

This raises important questions about:

- **Catastrophic forgetting** in fine-tuned SLMs
- **Multi-domain** fine-tuning strategies
- **Preserving general capabilities** while gaining specialization

Part – IV

Structured Workflows & Multi-Agent Systems

Monolithic prompts collapse when faced complexity with long problem descriptions with dozens of index sets, nested constraints, and domain-specific parameters.

Decompose the modeling task across **specialized agent roles**, domain patterns, and persistent state representations to achieve reliability.

Large-Scale Auto-Formulation via Structured Workflow

Liang, Lu, Mao, Sun, Yang, Zeng, Jin, Qin, Zhu, Teo — ***Focus: Enterprise-Scale Auto-Formulation Workflows***

The Industrial-Scale Problem

Standard LLMs fail catastrophically on complex, multi-page **industrial specifications**: dropping constraints, misassigning tensor indices, and conflating parameter scopes when given long, monolithic prompts.

Structured Extraction Workflow

1. **Schema-First Entity Extraction:** Sets, parameters, decision variables identified before any logic is written
2. **Atomic Logic Mapping:** Each business rule mapped to a single formal constraint clause
3. **Block Synthesis:** Constraint blocks assembled from atomic mappings
4. **Multi-Stage Validation:** Each stage verified before proceeding

Key Result

Standardizes high-dimensional tensor indexing and **eliminates dropped constraints** on complex enterprise specifications where direct-prompting baselines collapse entirely.

Large-Scale Auto-Formulation via Structured Workflow

Liang, Lu, Mao, Sun, Yang, Zeng, Jin, Qin, Zhu, Teo

Focus: Enterprise-Scale Auto-Formulation Workflows

LEAN-LLM-OPT

Handling industrial problems requires **decoupling entity/data schema design from logic formulation**. These are two separate cognitive tasks that must not be conflated within a single prompt.

Industry use case: **Singapore Airlines** Revenue Management

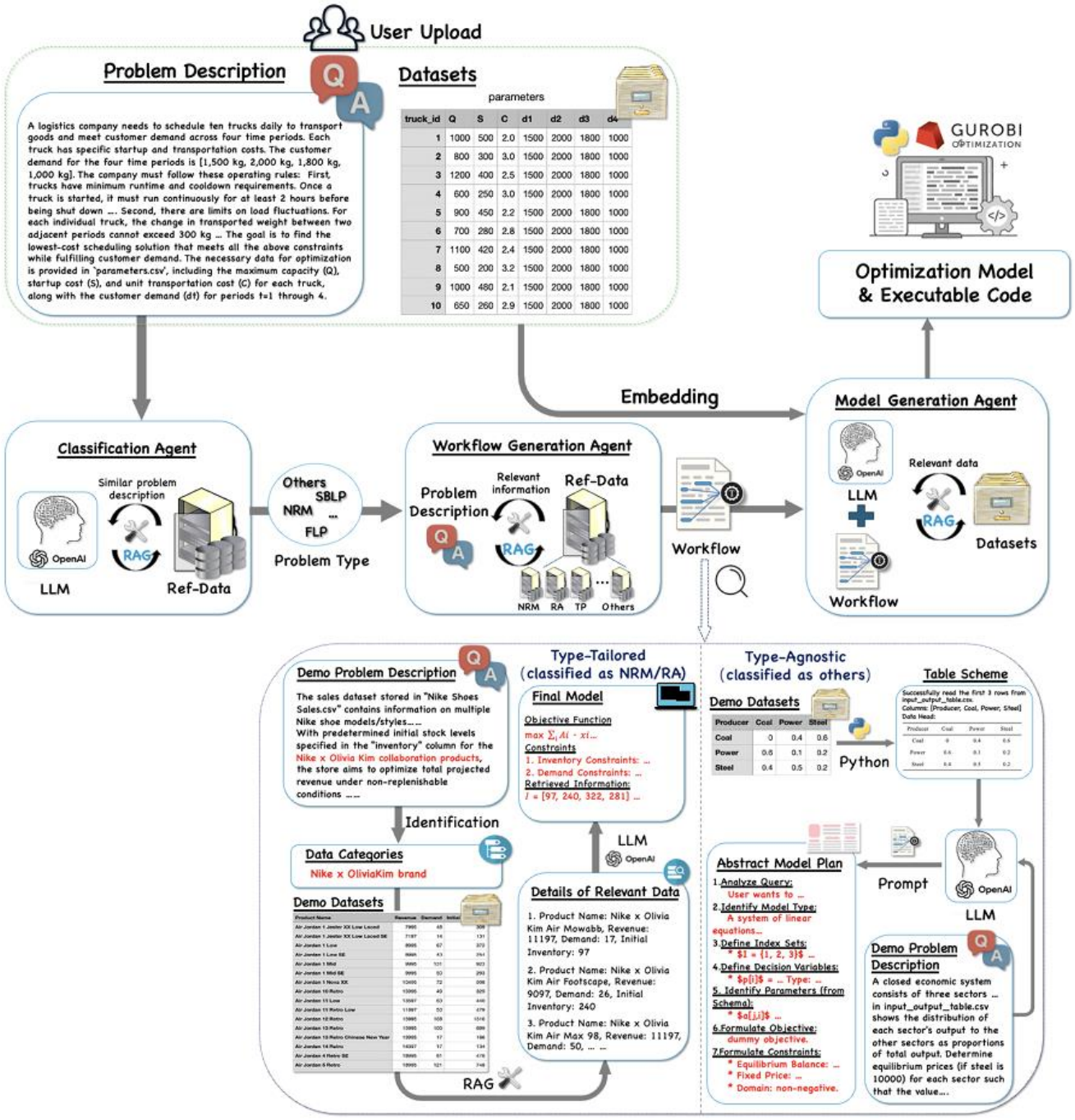


Figure 3 LEAN-LLM-OPT flow-diagram

OptiMUS-0.3: Modular Agentic State Management

AhmadiTeshnizi, Gao, Brunborg, Talaei, Lawless, Udell, 2024 — ***Focus: Agentic State Management & Modular Processing***

OptiMUS-0.3: Modular Agentic State Management

AhmadiTeshnizi, Gao, Brunborg, Talaei, Lawless, Udell, 2024 — ***Focus: Agentic State Management & Modular Processing***

Replaces naive single-prompt agent loops with a modular **multi-agent workflow** powered by **persistent State Management** via connection graphs and state JSON.

Modular Architecture Iterative Error Correction

Multi-agent system (parameter extraction, clause formulation, code generation) with **self-reflective error correction**, interactive feedback, and solver debugging.

Benchmark Dataset NLP4LP

Introduces 361 open-source optimization problems spanning easy, hard, and real-world INFORMS case studies.

Empirical Results

Outperforms standard GPT-4o by >43% on easy instances and >18% on hard instances, while reaching 42.9% on complex real-world cases

Detect Special Structures

Structure & Sifting Agents: Automatically detects **special structures** (e.g., SOS, indicator constraints) and implements constraint sifting to accelerate solver performance.

Core insight: **Context decomposition** via **persistent state graphs** is essential to prevent LLM performance degradation on lengthy industrial problem statements. Context overflow is a first-class failure mode.

Chain-of-Experts: Multi-Role OR Collaboration

Xiao, Liu, Astorga, van der Schaar, 2024 — ***Focus: Multi-Role Expert Collaboration & Process Decomposition***

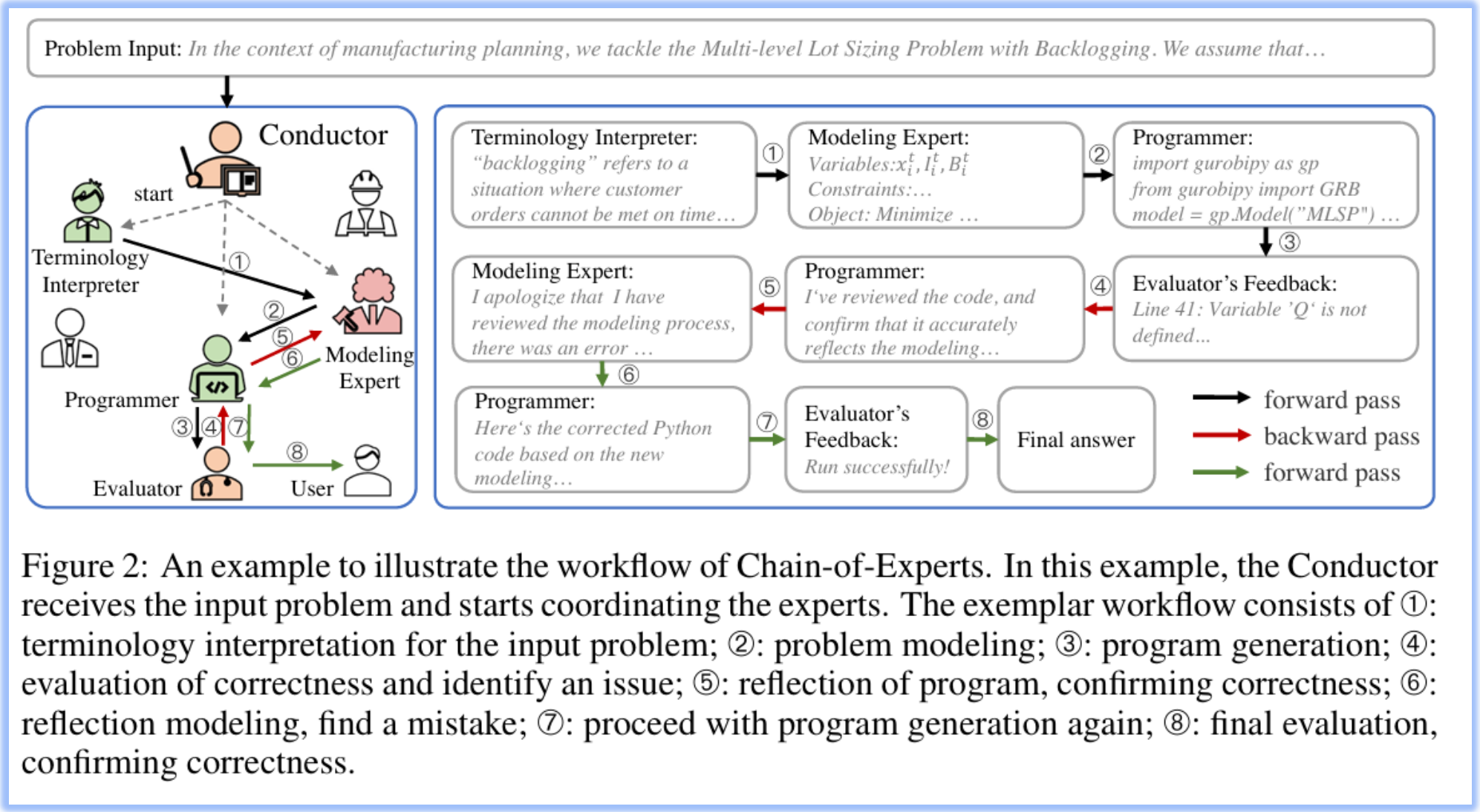


Figure 2: An example to illustrate the workflow of Chain-of-Experts. In this example, the Conductor receives the input problem and starts coordinating the experts. The exemplar workflow consists of ①: terminology interpretation for the input problem; ②: problem modeling; ③: program generation; ④: evaluation of correctness and identify an issue; ⑤: reflection of program, confirming correctness; ⑥: reflection modeling, find a mistake; ⑦: proceed with program generation again; ⑧: final evaluation, confirming correctness.

Breaking Down the Complexity Further

Current Limitations

Existing multi-agent approaches still inherit the **full problem complexity** rather than focusing on tractable sub-tasks.

Novelty

Gala centers around **global constraints**: high-level CP primitives like all_different that capture common patterns.

GALA: Global LLM Agents for CP Translation

Cai, Kadıoğlu, Dilkina — NeurIPS'2025 — ***Focus: Specialized Global Constraint Agents & Multi-Agent Collaboration***

Agentic meets Constraint Programming

all_different

Enforces distinct values across variables

cumulative

Enforces scheduling resource capacity over time

global_cardinality

Limits how many variables take each value

circuit

Creates a Hamiltonian circuit

Key Advantage

Gala aligns and **combines the key strength of Constraint Programming with Agentic Frameworks**, turning model generation into a collaboration of focused experts.

Global Constraints

High-level primitives that concisely **represent recurring patterns** such as distinct, resource limits, ordering, and counting found across planning, scheduling, assignment, and configuration problems.

Gala Agents

Each specialized agent **focuses only on detecting and encoding one constraint type**, simplifying the reasoning task dramatically.

Agentic Gala Framework

Gala is an agentic framework that addresses text-to-model translation with **global agents:** multiple specialized LLM agents **decompose** the modeling task by **global constraint type**.

Decomposition

Problem divided into smaller, well-defined sub-tasks.

Specialized Agents

Each agent detects and generates code for a specific class of global constraint.

Integration

Assembler agent integrates constraint snippets into complete model.

How Agents Work?

Specialized LLM Agents

LLM agent with a specialized prompt for each global. Each agent receives the full problem description, but its **instructions are local**: detect if the constraint is present and produce MiniZinc

Binary Classification

Each agent performs binary classification (constraint present or not) followed by code. The agent is instructed **not to produce any other modeling elements** beyond its constraint.

No Broader Modeling

Isolating each agent's focus, simplify the reasoning task. Unlike previous agents, **our agents do not need to understand the entire problem structure**, only whether a specific pattern appears.

> *"You are a MiniZinc modeling assistant specialized in detecting and modeling all_different constraints. Given a problem description, decide whether it requires one or more all_different constraints. If it does, generate only MiniZinc code specifying the all_different constraint and its variables. If it does not, return FALSE."*

The Assembler: Bringing It All Together

Assembler LLM takes over once constraint-specific agents return code snippets. Prompted as a MiniZinc modeler tasked with compiling a **complete** and **coherent** model.

01

Declare Variables

Define all decision vars and domains, renaming or merging for consistency

02

Analyze Constraints

Decide whether to include provided global constraints

03

Fill Gaps

Add remaining constraints from text not covered by hints

04

Define Objective

Determine optimization objective or satisfy goal

05

Finalize Model

Append solver boiler plate and output format

Key Benefit: Much of the heavy lifting is done by specialized snippets, the assembler focuses on gluing components together and writing boilerplate code.

Evaluation Framework

We conduct an initial evaluation of Gala, focusing on **two critical aspects** of the system's performance:

1 Global Agent Detection

The ability of global agents to correctly detect global constraints in problem descriptions.

2 End-to-End Performance

The end-to-end performance of the agentic pipeline compared to **baseline prompting strategies** and **chain-of-thought (CoT).**

Global Agent Detection Performance

We evaluate detection performance for seven global constraints using **GPT-OSS** on all 567 Text2Zinc instances

Constraint Type	Detection Rate (%)	False Detection Rate (%)
circuit	**100**	2.75
all_different	**88.1**	14.42
increasing	**78.6**	4.67
global_cardinality	**77.8**	17.43
lex_less	**71.4**	6.6
count	67.9	**28.3**
cumulative	58.3	17.59

Strong Detection Rates

Overall, our agents achieve detection rates around 70% to 80%, with circuit achieving perfect 100% detection.

Room for Improvement

False detection rates are generally low for rarer constraints. The main exception is `count (28.3%),` Distinguishing counting patterns from numerical constraints is a challenge.

Gala End-to-End Modeling Performance

We compare Gala with direct prompting (baseline) and COT on 110 verified Text2Zinc instances.

Model & Strategy	Execution Rate (%)	Solve Rate (%)
o3-mini Gala	**57.27**	**32.73**
o3-mini CoT	52.73	30.91
gpt-4o-mini Gala	**33.64**	**17.27**
gpt-4o-mini CoT	31.82	12.73
gpt-oss:20b Gala	**17.27**	8.18
gpt-oss:20b CoT	16.36	**10**
gpt-oss:20b baseline	11.81	7.27

Consistent Performance

Gala **consistently outperform** CoT on stronger models (o3-mini, GPT-4o-mini) across execution and solve

Decomposition Advantage

Gains come from agentic assembly with **no model tuning, prompt optimization**, or hyperparameter tuning.

Part – V

Search, Verification & Ensembling

Autoregressive generation can hallucinate constraints or get trapped in local logic errors with no recovery mechanism.

Reframes modeling as a **closed-loop search problem** using exact solver feedback as objective reward signals, surrogate pre-execution filtering to reduce compute, and ensemble strategies for robust model selection.

CP-Agent: Autonomous Agentic Constraint Programming

Szeider — Preprint, 2025 **Focus: Autonomous Agentic Frameworks for Constraint Satisfaction**

Reframes the entire CP modeling and solving lifecycle into an **end-to-end autonomous agent workflow** — moving beyond static code generation to full agentic problem-solving within a CP solver sandbox.

Problem Schema

ReAct Agent parses and structures the raw NL problem description into an internal model schema.

Model Generation

Synthesizes CP model code and executes it within a **solver sandbox**, capturing outputs and error traces. **Accessible via MCP**

Autonomously detects infeasibility signals and formulates **debugging hypotheses** without human intervention.

Error Tracing

Solver Execution

Revises constraint specifications and **re-executes iterating until a** valid solution is obtained.

Autoformulation via Monte Carlo Tree Search

*Astorga, Liu, Xiao, van der Schaar — **Focus: Search-Based Autoformulation & Closed-Loop MCTS Guidance***

Core Reframing

Model generation is reframed from feed-forward text translation into a **decision-making search problem** over the formulation space using **Monte Carlo Tree Search (MCTS)** to explore **alternative formulations** systematically rather than committing to a single autoregressive pass.

Architecture: LLM as Policy & Value Prior

- **LLM as policy/value prior:** Generates candidate formulation steps and scores their expected quality
- **Solver feedback as reward:** Syntax validity, satisfiability, and error logs serve as objective reward signals
- **Tree exploration:** Systematically backtracks from failed branches to explore alternative formulations

Why MCTS Beats CoT

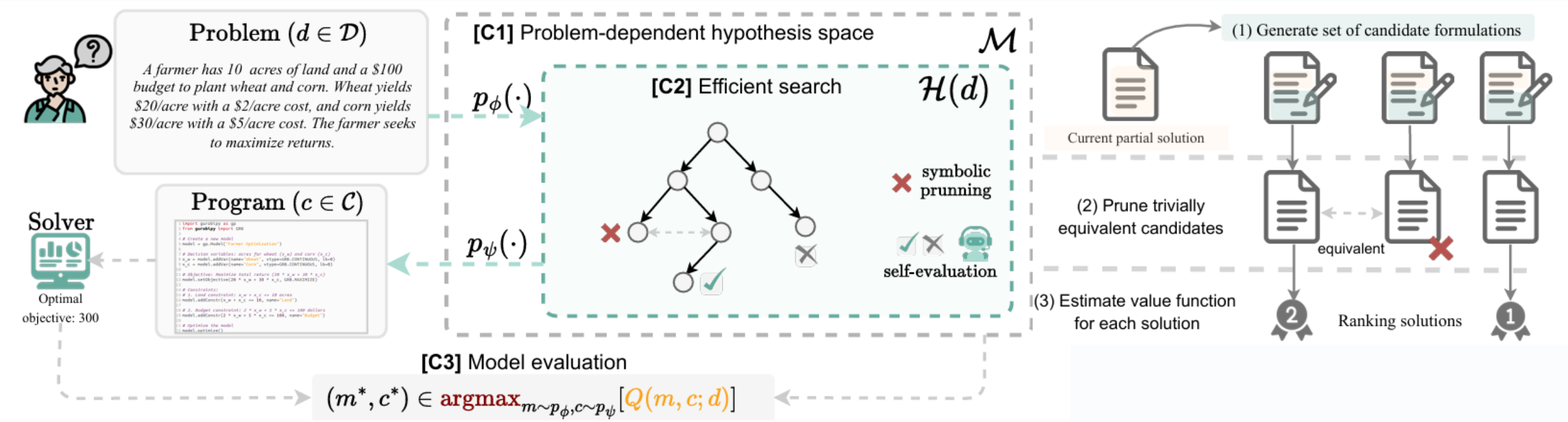
Single-pass Chain-of-Thought prompting **cannot backtrack** once an incorrect constraint is generated, it propagates forward and corrupts the entire model. **MCTS overcomes local minima and hallucination** by treating formulation as a search tree with verifiable intermediate states.

Takeaway

Combining probabilistic LLM generation with **exact solver feedback via MCTS** provides robust, verified model synthesis — a principled alternative to sampling with majority vote.

Autoformulation via Monte Carlo Tree Search

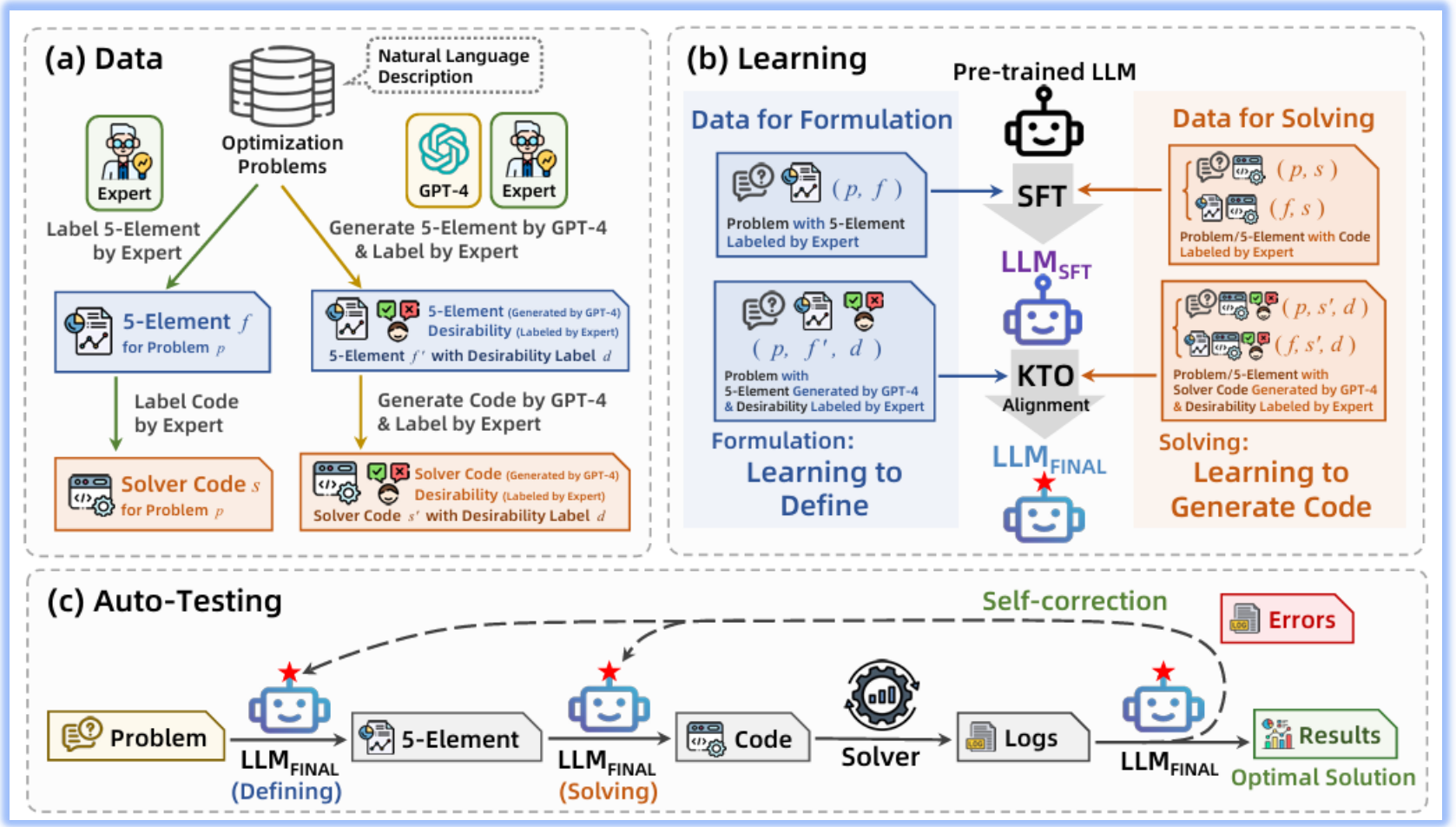
Astorga, Liu, Xiao, van der Schaar — ***Focus: Search-Based Autoformulation & Closed-Loop MCTS Guidance***



LLMOPT: End-to-End Formulation and Solving

Jian et al. — ICLR 2025 — ***Focus: End-to-End Learning Formulation & Solving Optimization***

- **Five-Element Formulation:** Introduces a structured intermediate representation **Sets, Parameters, Variables, Objective, and Constraints**
- **Multi-Instruction SFT:** Fine-tunes the base LLM on high-quality, expert-labeled datasets across two core tasks: **learning to define** (formulating the 5 elements) and **learning to solve** (generating executable solver code like Pyomo).
- **Kahneman-Tversky Optimization (KTO) Model Alignment:** Uses prospect theory-based preference alignment on expert-validated data to reduce LLM hallucinations
- **Auto-Testing with Self-Correction**



Part – VI

Human-in-the-Loop & Post-Solution Analytics

Finally, **multi-turn interaction** for discovery, refinement, and dialogue with candidate solutions.

Co-pilots must support **dynamic preference elicitation**, **counterfactual querying**, and plain-language post-solution **explanation** to earn stakeholder **trust**.

"I Want It That Way": Interactive Preference Elicitation

Lawless, Schoeffer, Le, Rowan, Sen, St. Hill, Suh, Sarrafzadeh, 2024 — ***Focus: Decision Support for CP + LLMs***

The Preference Construction Problem

Users rarely know all their constraints upfront. Real preferences are **discovered** and **refined dynamically** users see a candidate solution and then realize what they actually want.

MeetMate Architecture

Pairs LLM natural language understanding with a CP solver to continuously **elicit, parse, and translate** free-form user feedback into hard and soft CP constraints enabling iterative preference refinement

Validation

Evaluated on automated **meeting scheduling** via a diary study (*n*=64) and technology probe (*n*=10), demonstrating that interactive loops measurably boost agency, trust, and user satisfaction.

64

Diary Study

Participants in automated meeting scheduling evaluation

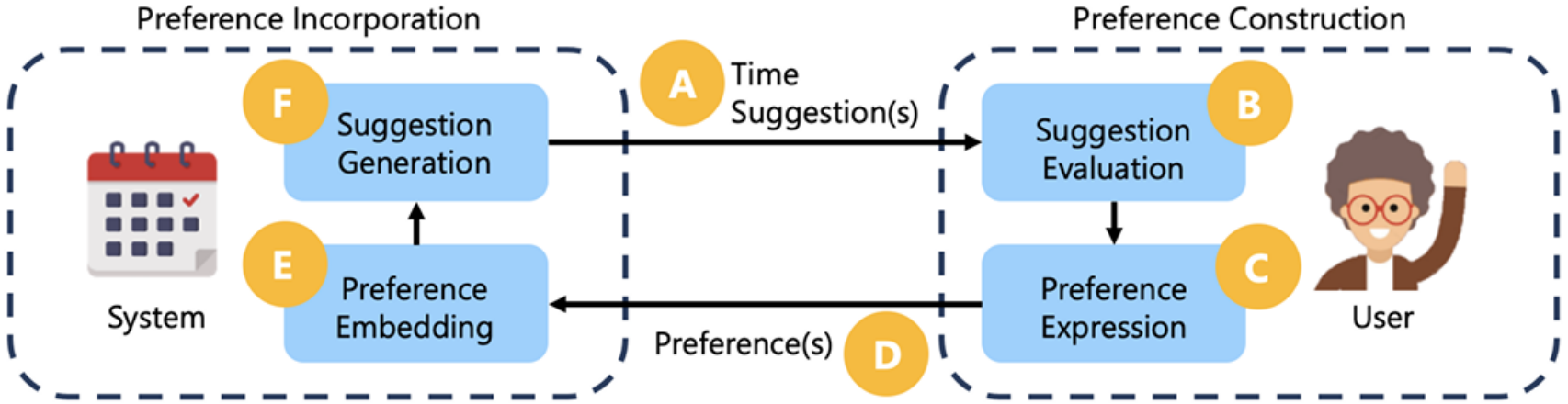
10

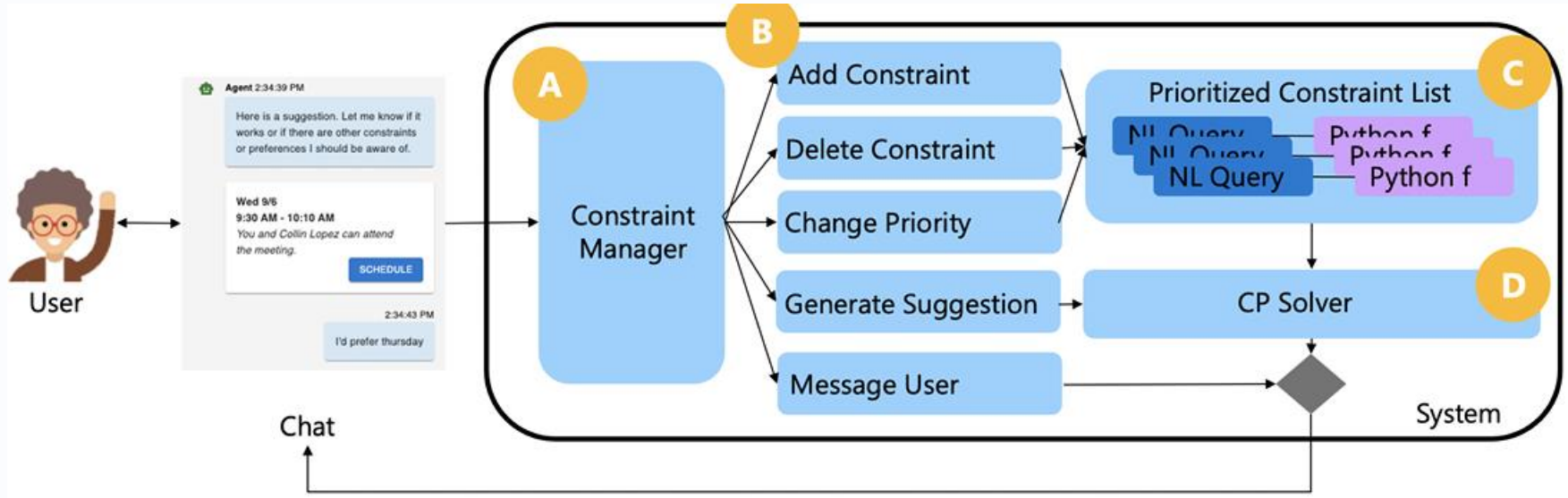
Technology Probe

In-depth qualitative participants confirming agency, trust, and satisfaction improvements

"I Want It That Way": Interactive Preference Elicitation

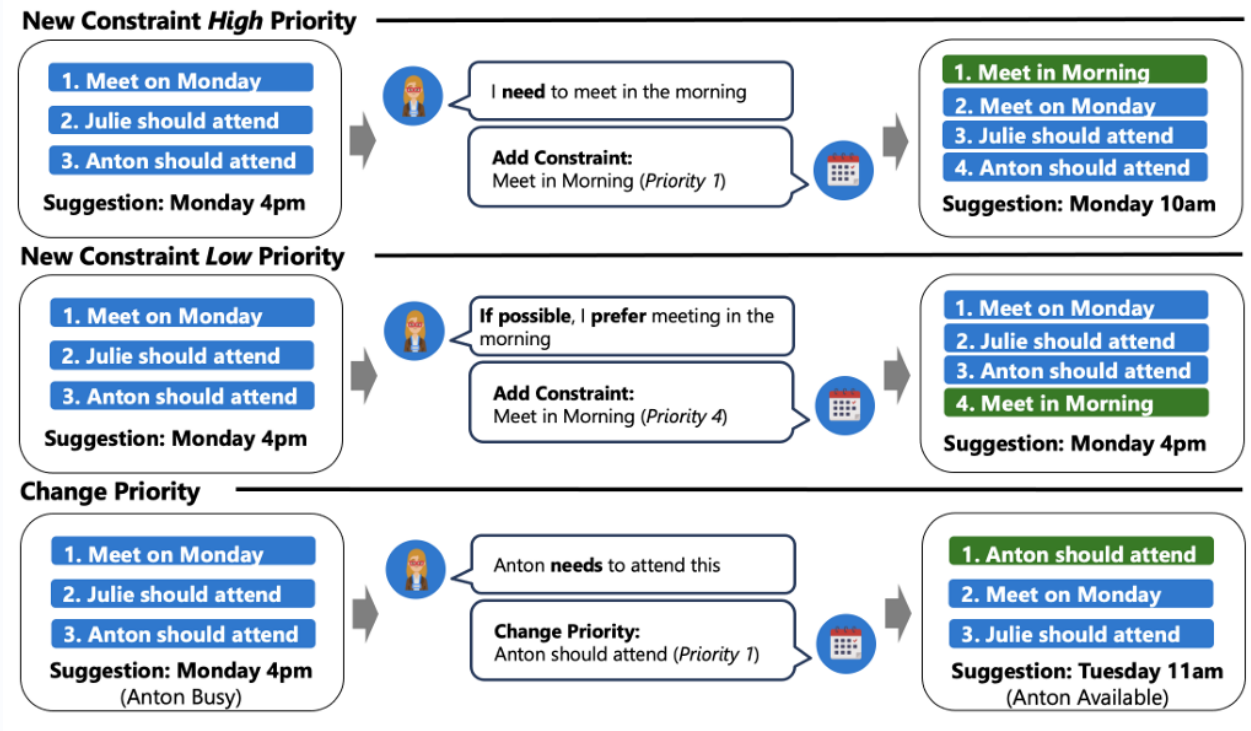
Lawless, Schoeffer, Le, Rowan, Sen, St. Hill, Suh, Sarrafzadeh, 2024 — ***Focus: Decision Support for CP + LLMs***

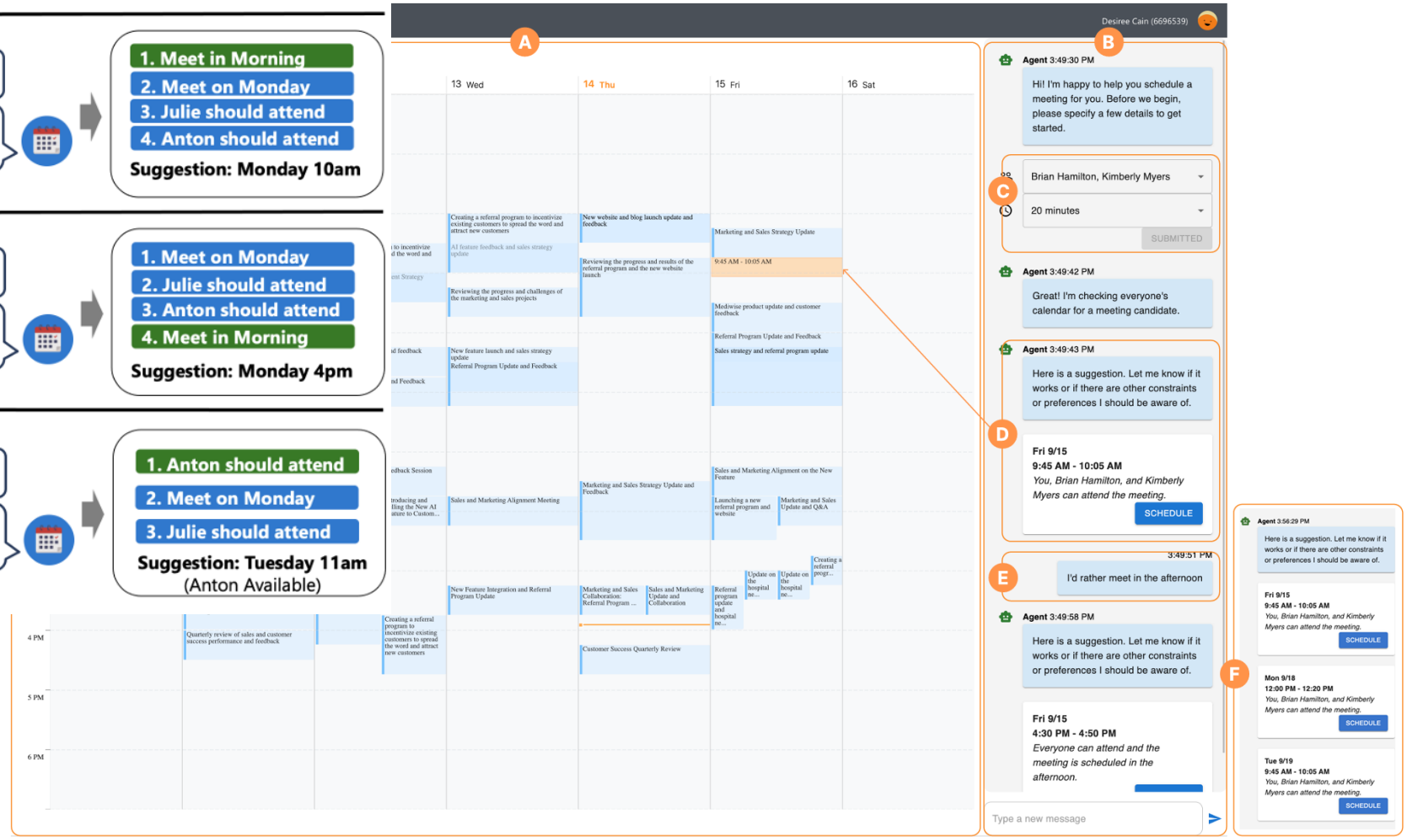




"I Want It That Way": Interactive Preference Elicitation

*Lawless, Schoeffer, Le, Rowan, Sen, St. Hill, Suh, Sarrafzadeh, 2024 — **Focus: Decision Support for CP + LLMs***





OptiChat: Bridging Models and Practitioners

Chen, Constante-Flores, Mantri, Kompalli, Ahluwalia, Li — **Focus: Model Introspection, Explanation & Post-Hoc Q&A**

Bridges the gap between raw solution vectors and business users by providing a **conversational Q&A interface** sitting directly on top of executed optimization models for post-hoc explanation and interactive what-if analysis.

Why / Why-Not Queries

Answers "**Why is** Facility A chosen?" and "**Why isn't** Route B used?" by querying active constraints, dual values, and **Irreducible Inconsistent Subsystems**, in plain language

Counterfactual What-If

Allows non-technical decision-makers to **test hypothetical scenarios** via natural language: "**What if** we add 20% capacity to Warehouse C?" without writing any solver code.

Stakeholder Trust Building

Makes the optimization model **inspectable** and **explainable** to domain analysts and executives

Co-pilots must extend **beyond model creation** to **support post-solution explanation**, introspection, and stakeholder trust building.

Roadmap Covered

Our roadmap covers **six research themes** from position papers vision to operational aspects. Each group builds on the last, forming a **coherent** agenda for next-generation Modeling Copilots.

Vision & Position Papers

Define the paradigm shift from solver-centric to human-centric modeling interfaces leveraging LLMs.

Foundations & Performance Baselines

Extract mathematical entities from text and establish diagnostic evaluation protocols.

Fine-Tuning & Synthetic Data Engines

Build verifiable data pipelines to adapt open-weights SLMs and fine-tuning for declarative modeling.

Structured Workflows & Multi-Agent Systems

Decompose complex specifications across specialized agent roles and persistent state.

Search, Verification & Ensembles

Frame formulation as a closed-loop search problem with exact solver feedback.

Human-in-the-Loop & Post-Solution Analytics

Support multi-turn preference elicitation, counterfactual querying, and explanation.

Session B.2

Conclusions & Outlook

Presenter: Tias Guns

Tutorial Agenda

Session A: Introduction, data, evaluation (9:00 - 10:30)

A1) Introduction (30 min): *Tias Guns*

A2) Benchmarking & Evaluation (60 min): *Dimos Tsouros*

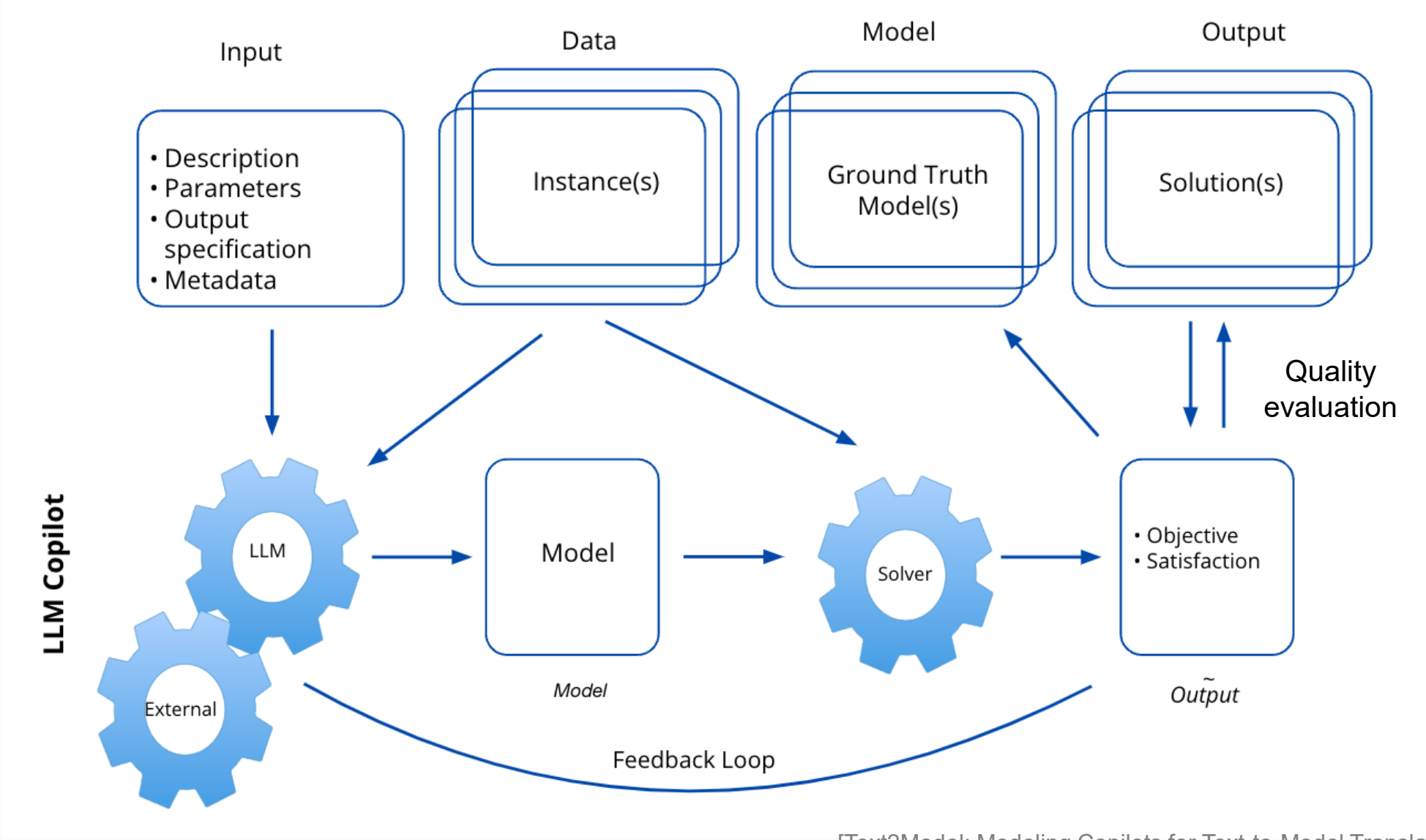
Break (10:30 - 11:00)

Session B: Modelling copilots (11:00 - 12:30)

B1) Modelling copilots (60 min): *Serdar Kadioglu*

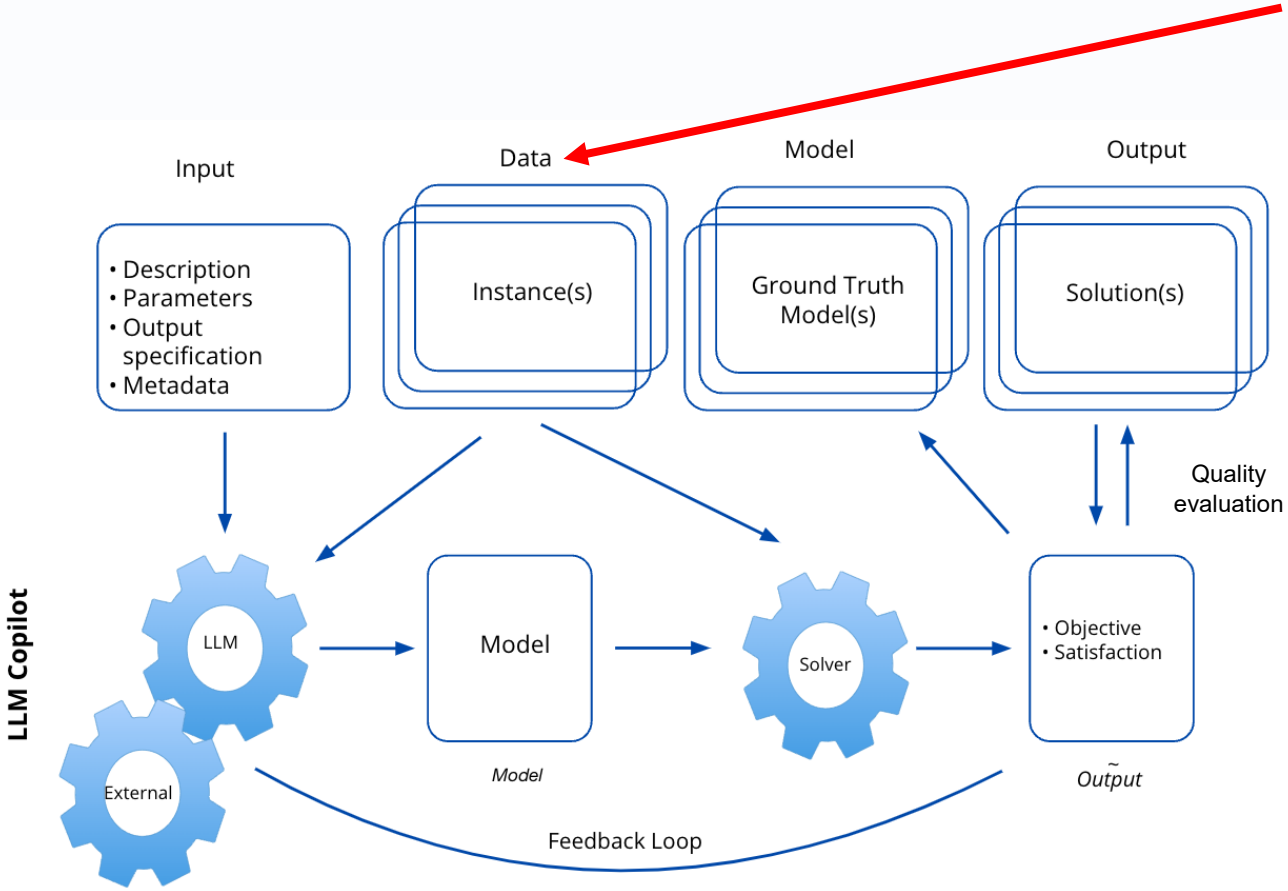
B2) Wrap-up and outlook (30 min): Tias Guns

Modeling CoPilots



[Text2Model: Modeling Copilots for Text-to-Model Translation, S. Kadioglu et. al.]

Modeling CoPilots, challenges

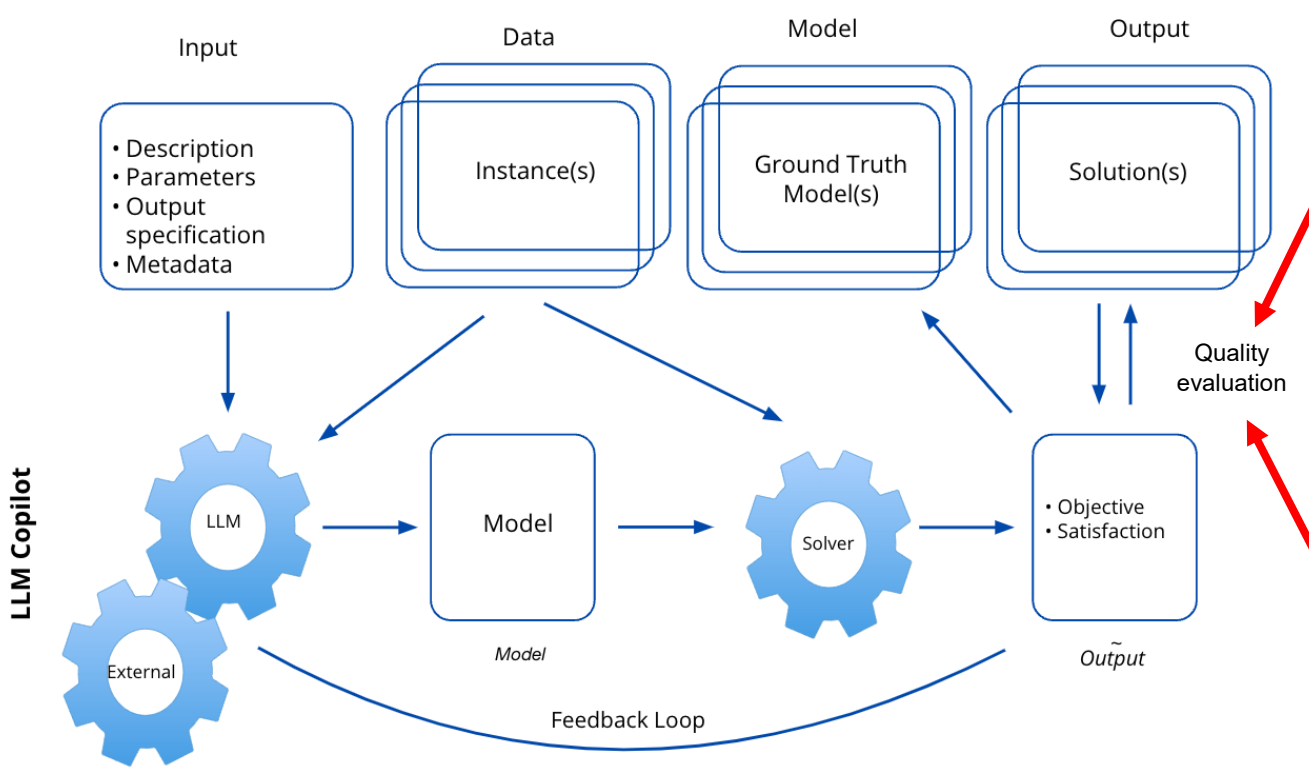


Scaling to Industrial Problem Graphs & Dynamic Data

Co-pilots that jointly reason over **multi-relational database schemas**, graph architectures, and optimization formulations automatically generating data-binding layers (SQL/OR-map) alongside model

[Text2Model: Modeling Copilots for Text-to-Model Translation, S. Kadioglu et. al.]

Modeling CoPilots, challenges



Formal Semantic Verification & Hallucination Guarantees

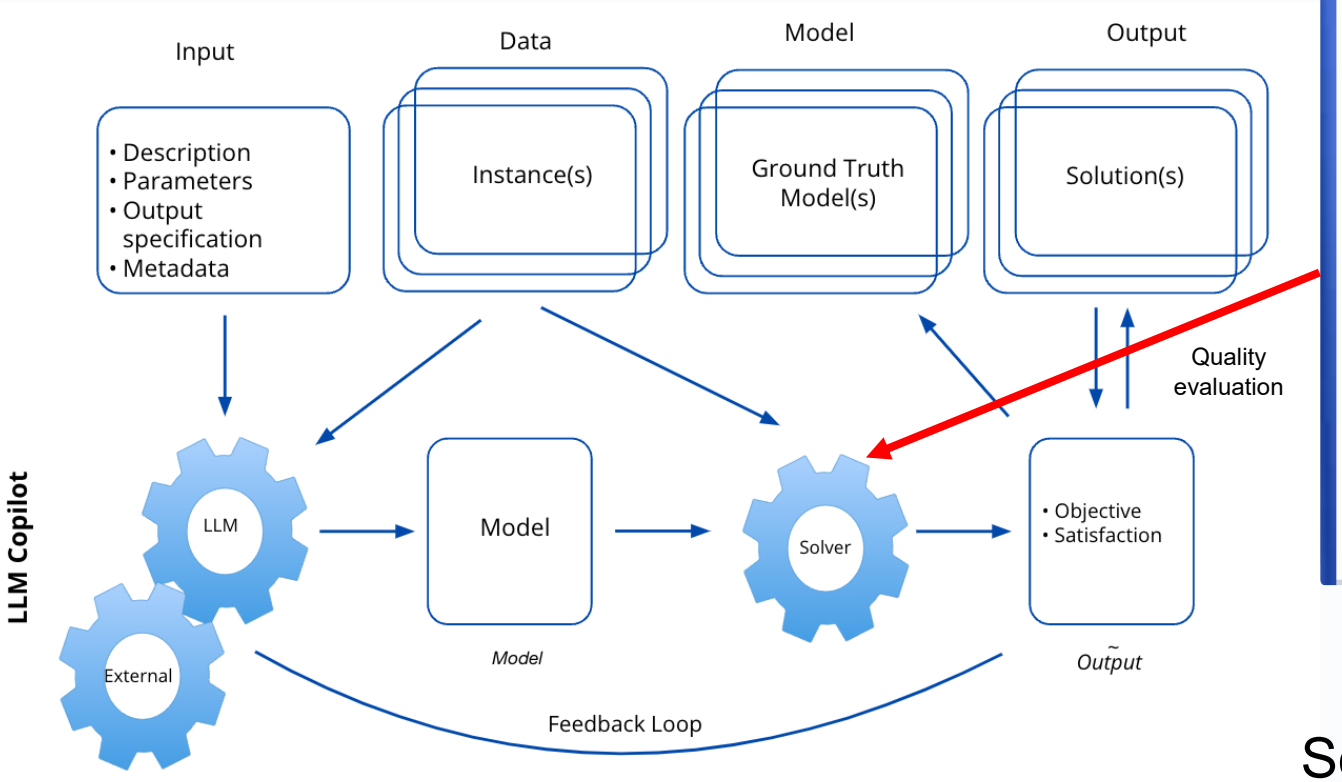
Formal **equivalence checkers**, **property-based testing suites**, and **dynamic counter-example generators** that prove semantic parity between natural language specifications and mathematical models.
Formal correctness & Lean...

Beyond correctness: **efficiency**

How **quickly** a model can find **good or optimal** solutions; evaluated at different time-points? On diverse, small and large instances?
And also given small and large instances as input?

[Text2Model: Modeling Copilots for Text-to-Model Translation, S. Kadioglu et. al.]

Modeling CoPilots, challenges



Solver frameworks can matter

MiniZinc is a low-resource language, on DCP-Bench Python based solvers had higher accuracies.

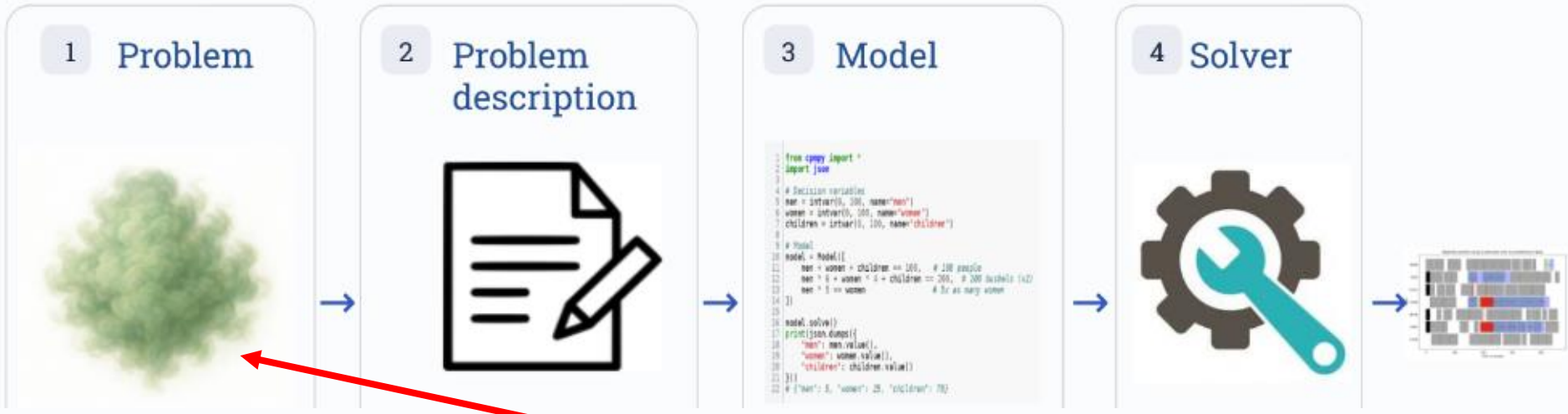
Many papers only use one solver.

General-purpose AI systems should be able to use **any** suitable solver?

Solver-agnostic DCP-Bench-Open zoo: https://dcp-bench.github.io/DCP-Bench-Open/problems/abbots_puzzle.html

[Text2Model: Modeling Copilots for Text-to-Model Translation, S. Kadioglu et. al.]

What is the starting point?



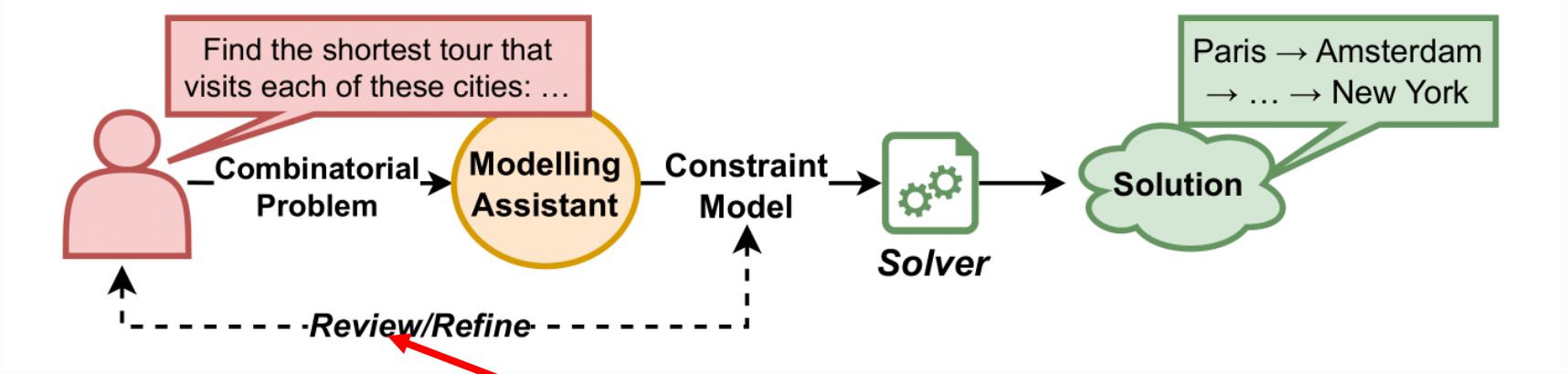
?

- **Benchmarks** Many problem statements were underspecified or ambiguous.
- **NL → model ambiguity** Most NL-to-model approaches assume the 'problem description' is an unambiguous and complete spec

Conversational Reformulation & Active Disambiguation?

Active-learning co-pilots that detect **ambiguity**, compute the information value of clarification, and **proactively ask low-friction targeted questions** (e.g., "Is this overtime limit a hard constraint or a soft penalty?").

Modeling CoPilots, human-in-the-loop

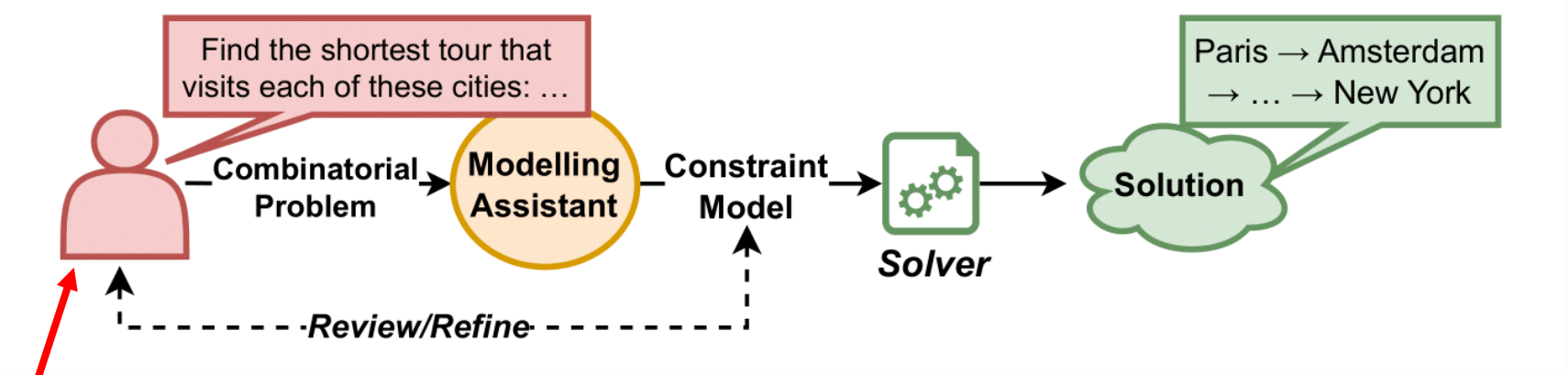


Conversational Reformulation & Active Disambiguation

Active-learning co-pilots that detect **ambiguity**, compute the information value of clarification, and **proactively ask low-friction targeted questions** (e.g., "Is this overtime limit a hard constraint or a soft penalty?").

What benchmarks and how to evaluate?

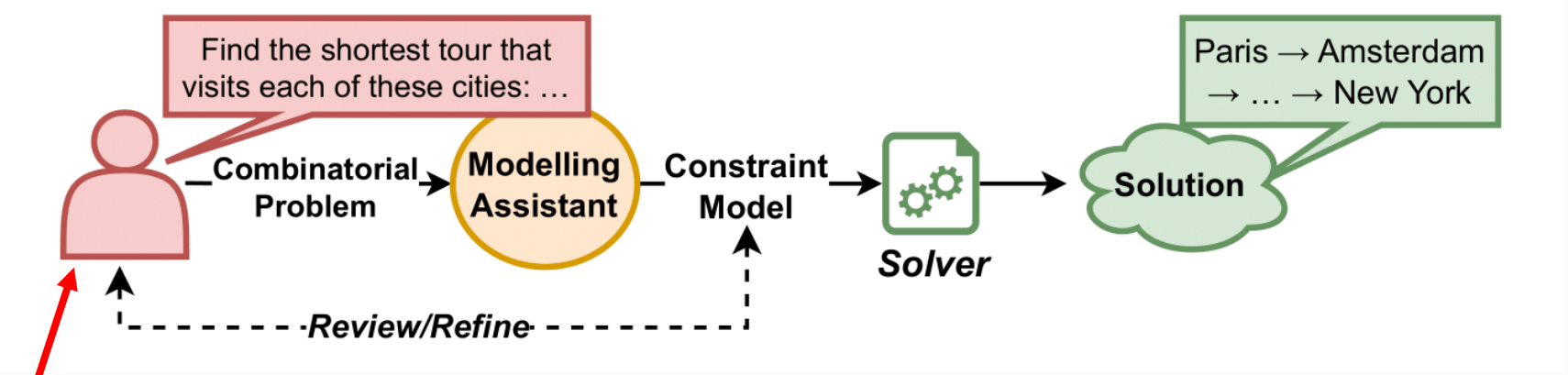
Modeling CoPilots, human-in-the-loop



WHICH human?

- **Students** They are there to learn a language, on academic exercises.
- **IT at a non-optimisation company** They want the problem solved, behind the scenes.
- **End users** They want to discover the problem, model it, and change it, in one environment.
- **Professional modellers** They want a faster first model, then to improve it, then to change it when the client does.

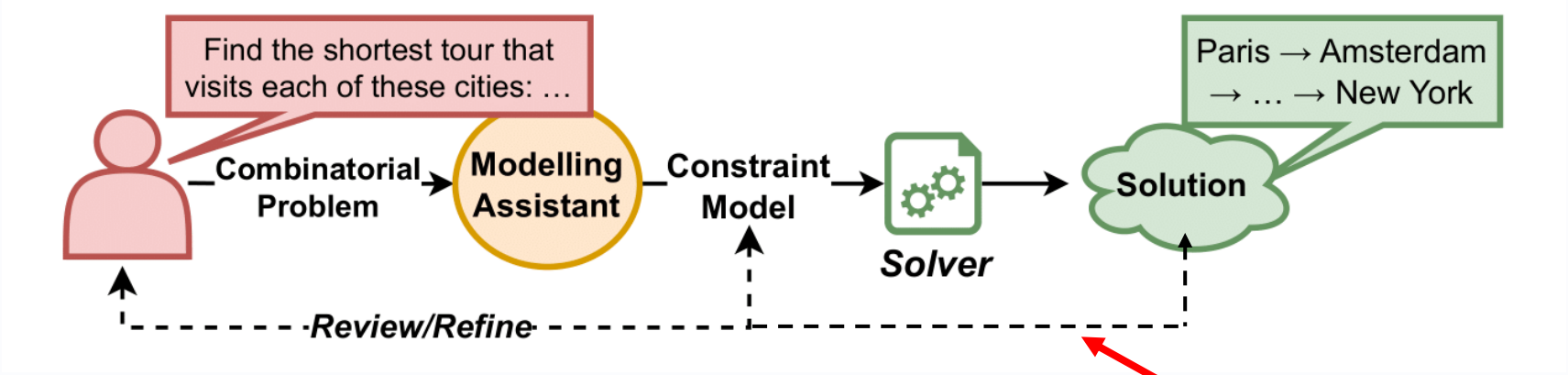
Modeling CoPilots, human-in-the-loop



Only a single user?

- **Multi-user** where users can make mistakes or disagree: incompatibility resolution
- **Conflicting preferences** would require elicitation, mediation and conflict resolution
- Limited to small-scale studies, or how to do systematic benchmarking and large-scale evaluation?
  => Requires closer ties to Human-Computer Interaction and Social Science researchers?

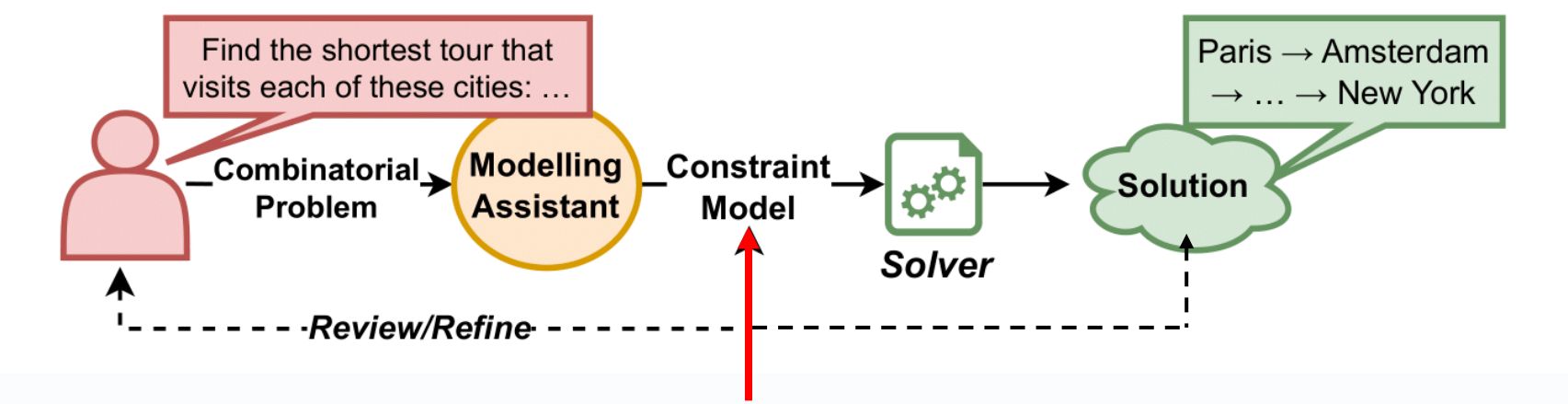
Modeling CoPilots, human-in-the-FULL-loop



End-to-End Explainability, IIS Debugging & Counterfactual Reasoning

Autonomous **post-solution analytics** translating infeasibility proofs, shadow prices, and sensitivity analysis into executive counterfactual narratives (e.g., "Increasing budget by $5k resolves bottleneck at Fac B").

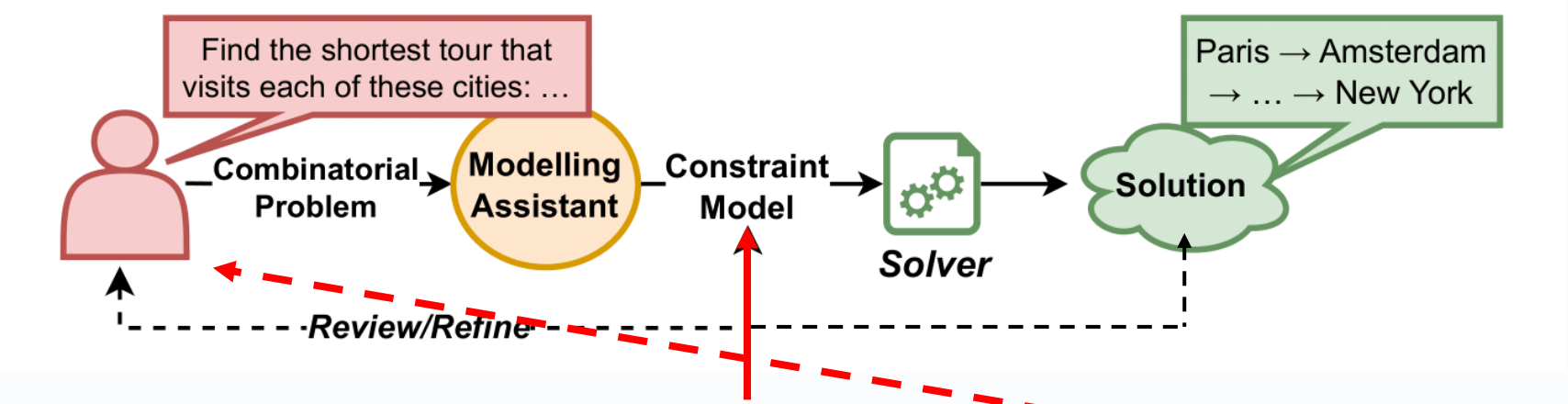
Modeling CoPilots, human-in-the-FULL-loop



If everything is natural language, why still have a formal model?

- **Pro:** You can check it, reuse it on any new data, change it, and sometimes prove UNSAT/optimality
- **Contra:** High-quality custom algorithms can scale to very large problems
- **Hybrid:** Let agents freely combine different solving strategies including solvers and custom algorithms...

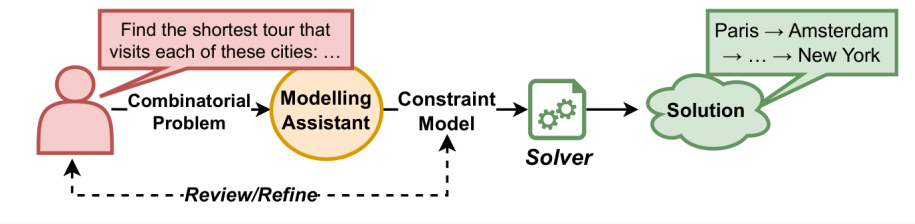
Modeling CoPilots, human-in-the-FULL-loop



Does the modeling language/solver still matter? **Also depends on the end-user!**

- **Students** pedagogical language chosen by teacher, learns 1 system.
- **IT at a non-optimisation company** might prefer easy to install and maintain?
- **End users** cares more about the user-experience, different libraries might support this better/worse
- **Professional modellers** Often want to compare multiple systems and have fine-grained control

When the problem changes

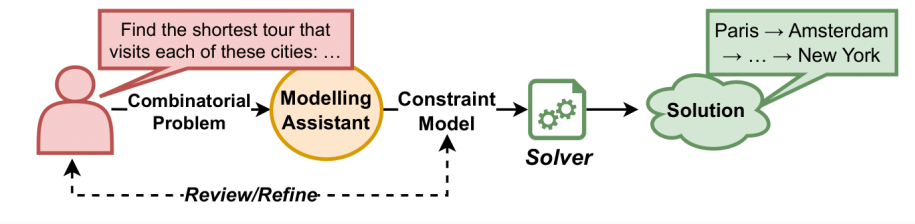


- **Next week** More data, or a new hard constraint from the client.
- **A model you can edit** A specialised algorithm is frozen. That is why you still want a model, not only a search routine.

Knowledge from historic solutions

- **In the organisation** Organisations have old rosters. Preferences often hide in past solutions.
- **Regenerating from English** Regenerating from English every Monday throws that away. Better to update the model, or reweigh. ChatOpt: preferences from historic solutions, and concept drift.

LARGE-scale problems…



- **One generated model** The generated model is one piece. It can be correct and still time out.
- **Decomposition is modelling** Many decomposition techniques into a ‘master’ problem and one or many sub-problems (benders, column generation, repair-and-destroy, LNS)
  → able to model both? Multi-solver or other hybredisations?
- **Long industrial specs** Entity schema first? Or raw excell sheets, data exports…
- **Optimisation under uncertainty** Model both the uncertainty and the optimisation?

  Sometimes combinatorial modeling is half-data science, only half optimisation;
  evaluate these skills separately or together? With what tools/libraries/evaluation frameworks?

Summary: open problems

- **Correctness** Solution accuracy, single/multi instance, equivalence…
- **Efficiency, scale, decomposition** Correct, and still usable at a realistic size. Decomposition is part of that.
- **Hybrid solving** Warm-starts and neighbourhoods; custom search in parallel to the solver
- **Multi-user and conflicting preferences** Several stakeholders, preferences that conflict, reconciliation…
- **Data-rich and uncertain applications** Requiring data analytics and derived data representations, or uncertainty and the need to model that
- **New modelling languages?** If an LLM writes it and a person checks it, what belongs in the language?

Slides at the Tutorial website. See also material, and other events, workshops, tutorials, special issues…

https://sites.google.com/view/ijcai26-llm-con

Cite as:

Guns, T., Tsouros, D., & Kadioglu, S. (2026, August 17). LLMs for Constraint Modeling, IJCAI26 tutorial. Zenodo. https://doi.org/10.5281/zenodo.21998210

```
@misc{guns_2026_21998210,
  author       = {Guns, Tias and
                  Tsouros, Dimosthenis and
                  Kadioglu, Serdar},
  title        = {LLMs for Constraint Modeling,
IJCAI26 tutorial},
  month        = aug,
  year         = 2026,
  publisher    = {Zenodo},
  doi          = {10.5281/zenodo.21998210},
  url          =
{https://doi.org/10.5281/zenodo.21998210},
}
```