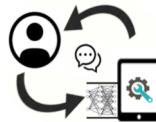


Constraint Modelling with LLMs Using In-Context Learning

Kostis Michailidis, Dimos Tsouros, Tias Guns



KU LEUVEN

Constraint Modelling: An Example

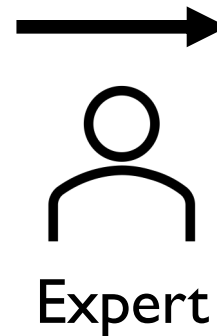
Context: A popular actor with many offers.

Requirement 1: Only one movie per day.

Requirement 2: Maximize income.

Which movies to choose?

	Movie Title	Start Day	End Day
0	Tarjan of the Jungle	4	13
1	The Four Volume Problem	17	27
2	The President's Algorist	1	10
3	Steiner's Tree	12	18
4	Process Terminated	23	30
5	Halting State	9	16
6	Programming Challenges	19	25
7	Discrete Mathematics	2	7
8	Calculated Bets	26	31



Decision Variables:

$movies_0, movies_1, \dots, movies_8$

Domain:

$movies_i \in \{0, 1\}$

Constraints:

- $movies_0 + movies_2 \leq 1$
- $movies_0 + movies_3 \leq 1$
- $movies_0 + movies_5 \leq 1$
- [...]
- $movies_4 + movies_8 \leq 1$

Objective:

$$\max\left(\sum_{i=0}^8 movies_i\right)$$

Constraint Modelling: An Example

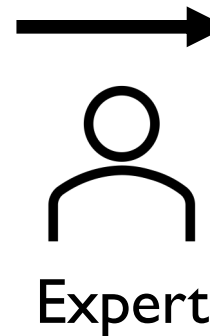
Context: A popular actor with many offers.

Requirement 1: Only one movie per day.

Requirement 2: Maximize income.

Which movies to choose?

	Movie Title	Start Day	End Day
0	Tarjan of the Jungle	4	13
1	The Four Volume Problem	17	27
2	The President's Algorist	1	10
3	Steiner's Tree	12	18
4	Process Terminated	23	30
5	Halting State	9	16
6	Programming Challenges	19	25
7	Discrete Mathematics	2	7
8	Calculated Bets	26	31



```
1 from cpmPy import *
2
3 # Data (title, start, end)
4 movies = [["Tarjan of the Jungle", 4, 13],
5           ["The Four Volume Problem", 17, 27],
6           ["The President's Algorist", 1, 10],
7           ["Steiner's Tree", 12, 18],
8           ["Process Terminated", 23, 30],
9           ["Halting State", 9, 16],
10          ["Programming Challenges", 19, 25],
11          ["Discrete Mathematics", 2, 7],
12          ["Calculated Bets", 26, 31]]
13 num_movies = len(movies)
14
15 # Decision Variables
16 selected_movies = boolvar(shape=num_movies) # 1 if the movie is selected, 0 otherwise
17
18 # CP Model
19 model = Model()
20
21 # Non-overlapping movie schedules
22 for i in range(num_movies):
23     for j in range(num_movies):
24         # Check if the intervals overlap for each pair of movies
25         if i != j and movies[i][2] > movies[j][1] and movies[j][2] > movies[i][1]:
26             # Constraint: the movies cannot be selected together
27             model += ~(selected_movies[i] & selected_movies[j])
28
29 # Objective: Maximize the number of selected movies
30 model.maximize(sum(selected_movies))
31
32 # Solve
33 model.solve()
34 print(selected_movies.value())
35 # Output:
36 # [False False False False False True True True True]
```

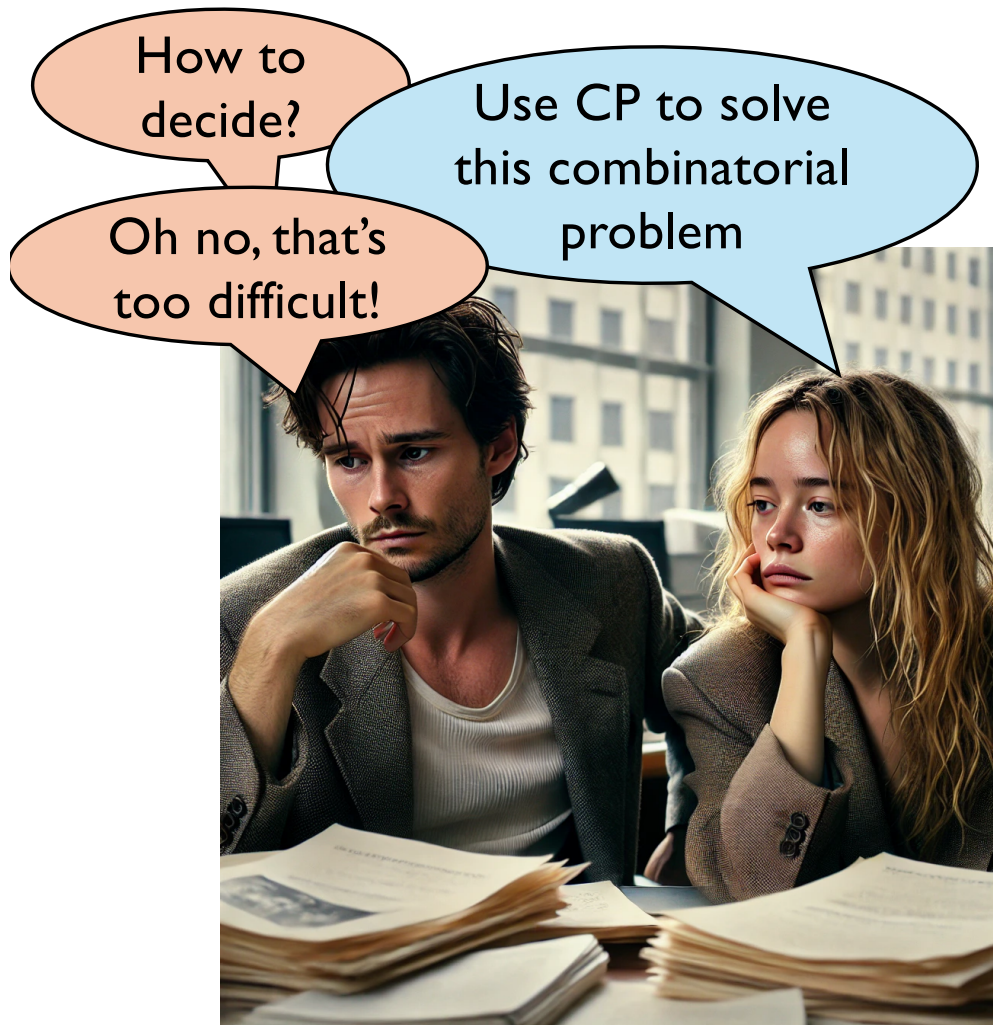
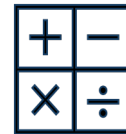


Image created using OpenAI's ChatGPT.

CP is **powerful**, but modelling requires **expertise** in:



Domain Knowledge



Mathematical Skills



Solver Formalisms

Making CP More Accessible

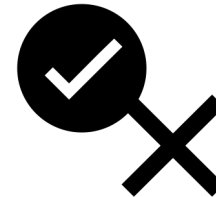
Modelling Tools



CPMPy

Constraint Acquisition

Users provide or classify (non-)solutions



From Natural Language

Textual descriptions of combinatorial problems

Make a CP model
for me, please!

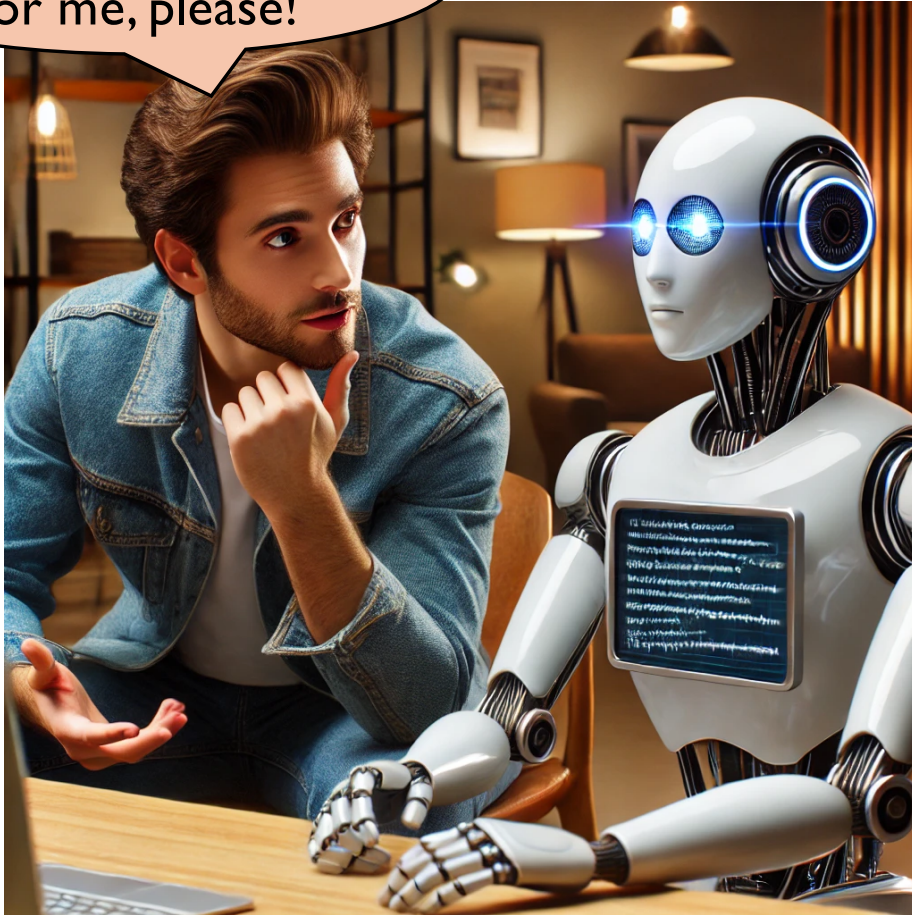
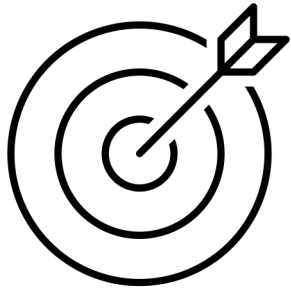
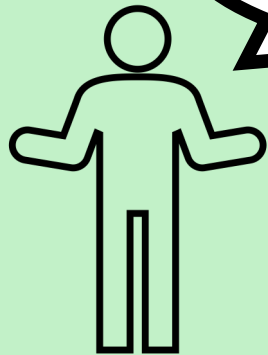


Image created using OpenAI's ChatGPT.

Can
Large Language Models
assist in
Constraint Modelling?



An LLM-based *Constraint Programming & Modelling* **assistant** for combinatorial problems given only their natural language description



We're a team of 10, and we need to complete 5 tasks in the shortest time possible.

Solution Found!

```
from cpmPy import *

# Define the number of workers and tasks
num_workers, num_tasks = 10, 5

# Time taken for each worker to complete each task
times = [[...], ...]

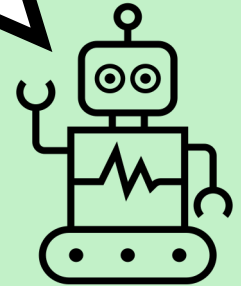
# Decision variables
X = boolvar(shape=(num_workers, num_tasks)) # Assignment matrix

# Constraints
model = Model([
    # Each worker can only be assigned to at most one task
    [sum(X[i]) <= 1 for i in range(num_workers)],
    # Each task must be assigned to exactly one worker
    [sum(X[:, j]) == 1 for j in range(num_tasks)],
])

# Objective: Minimize the total time taken
model.minimize((X * times).sum())

model.solve()
print(X.value())
```

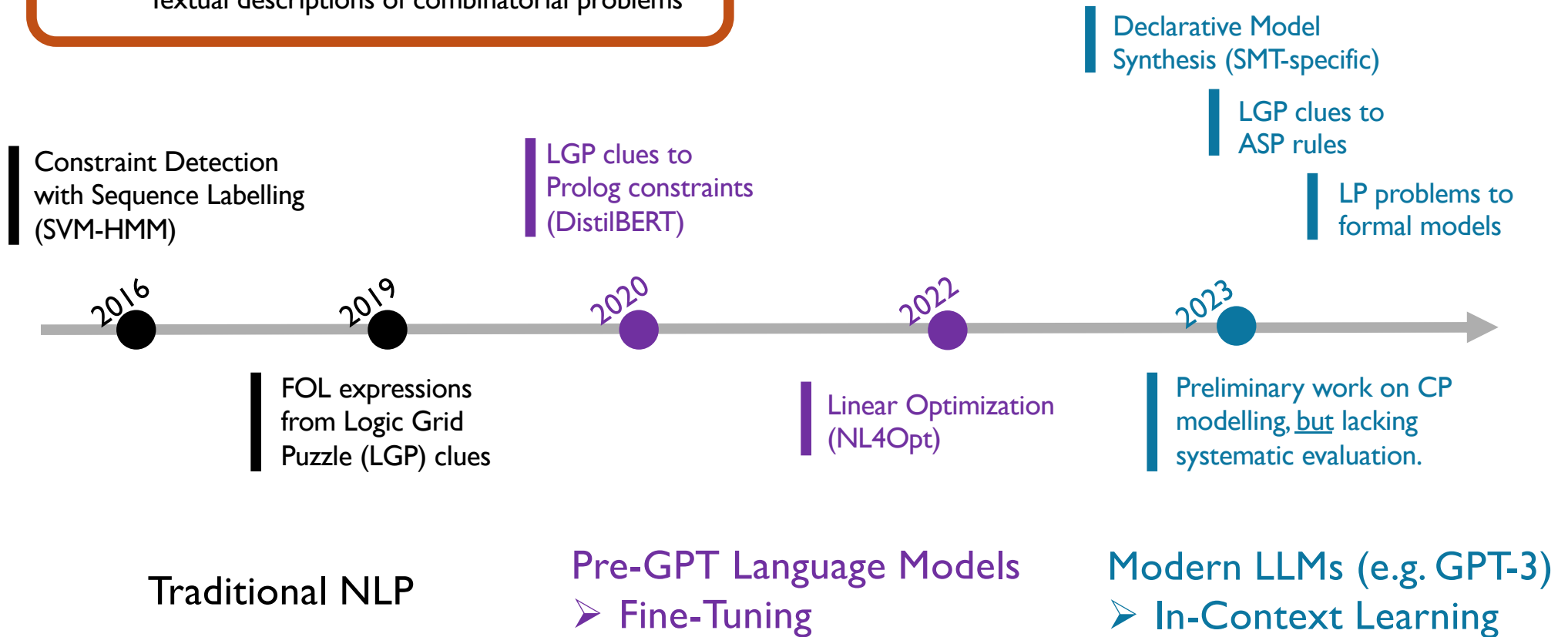
Here is an executable model in Python:





From Natural Language

Textual descriptions of combinatorial problems



Background: Large Language Models

$$LLM(Prompt) = Response$$

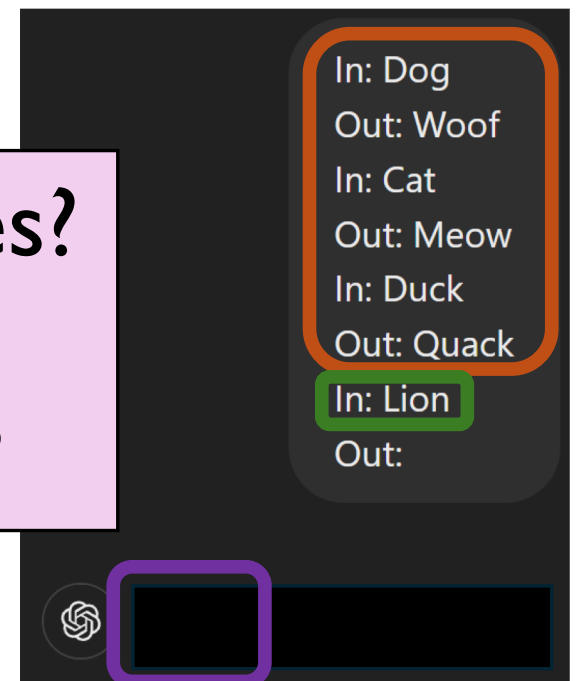
In-Context Learning (ICL):

An emergent ability of LLMs ($\geq$ GPT-3).

- Task Learning:

$$LLM(\boxed{C} \oplus \boxed{Q}) \approx \boxed{R}$$

1. What examples?
2. How many?
3. In what order?



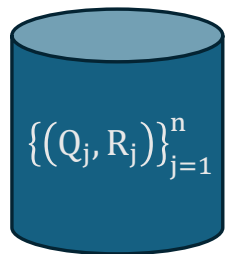
<https://chatgpt.com/share/431e9c84-1fe4-412e-aa91-1f9f00063e9d>

How to choose C ?



Baseline: Static/Random

Retrieval-Augmented ICL

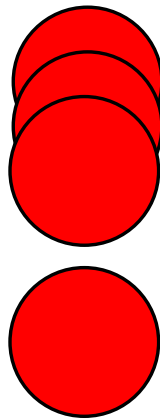


C

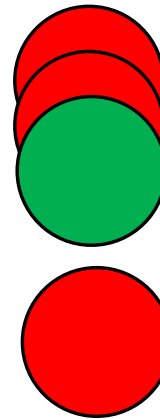
C

Q

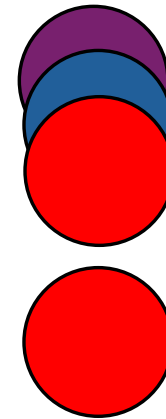
Similarity



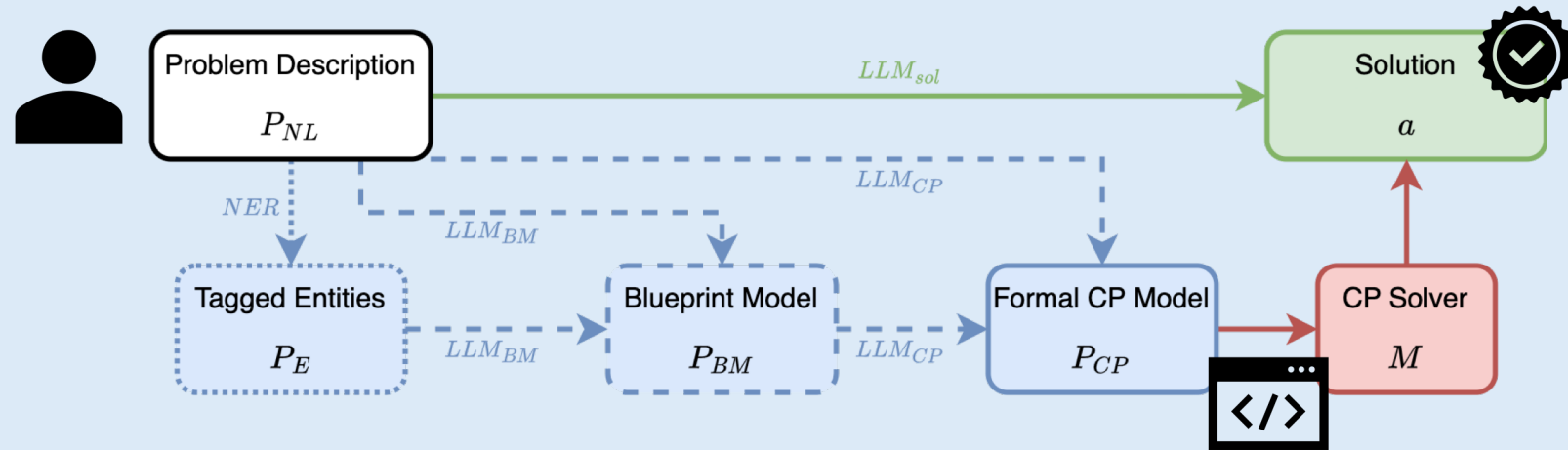
Diversity



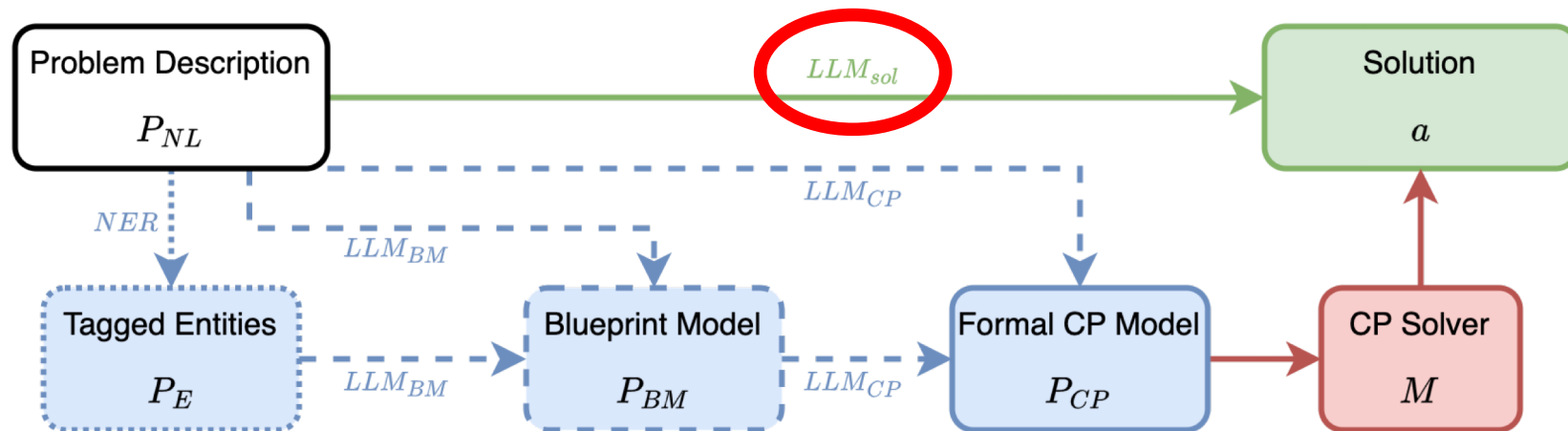
Recency



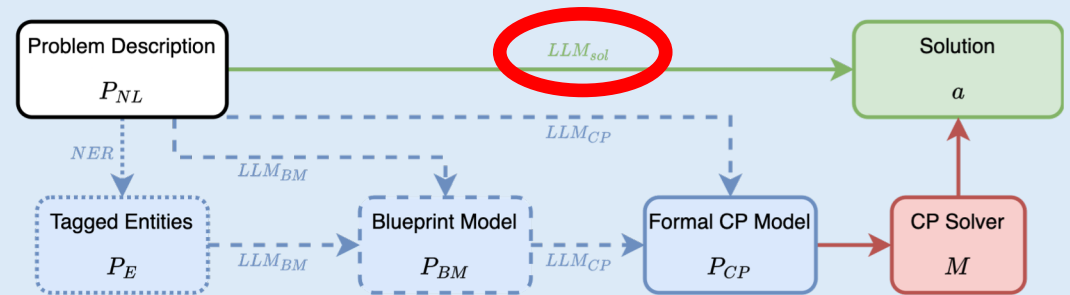
Using LLMs as assistants for solving constraint problems



I. LLMs as solvers



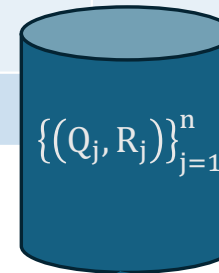
I. LLMs as solvers



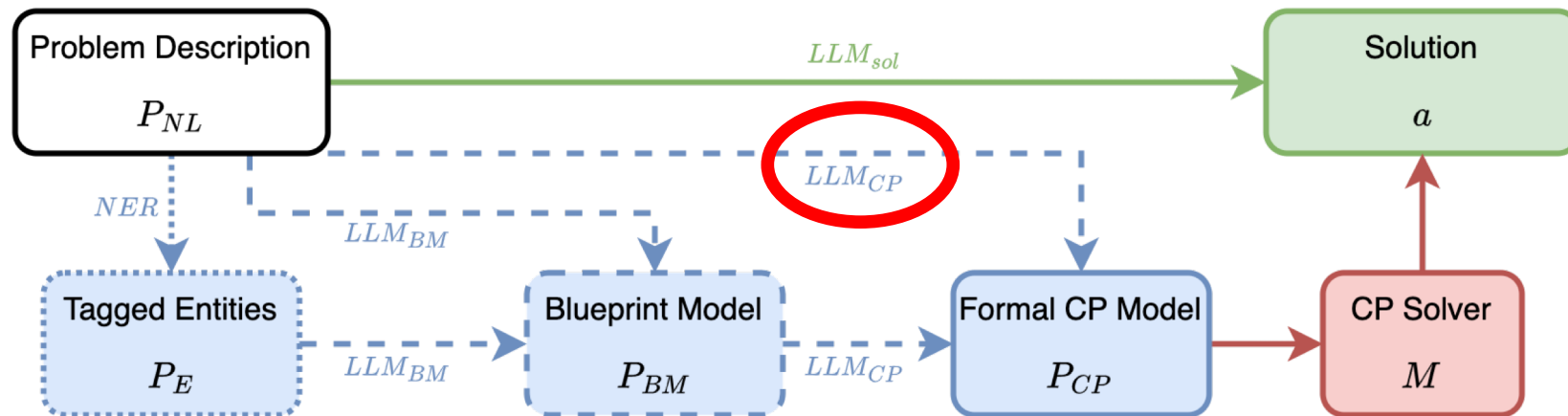
Solution generated with
Chain-of-Thought

$$a = LLM(C \oplus Q)$$

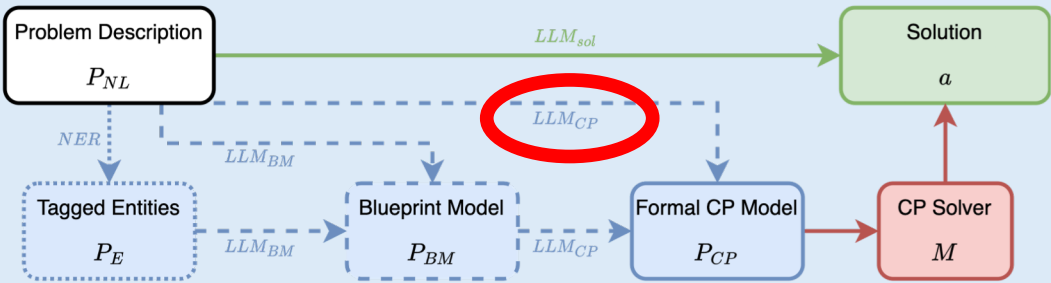
j	Q_j (Problem Desc)	R_j (CoT solution)
1	A popular actor... Which movies to choose?	Let's think step by step... Thus, the final solution is: $\{movies_0: 0, movies_1: 0, \dots\}$
...	...	...
n	...	...



2. LLMs as modellers



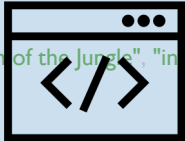
2. LLMs as modellers

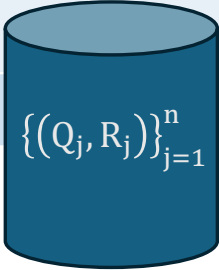


Generation of CP models:

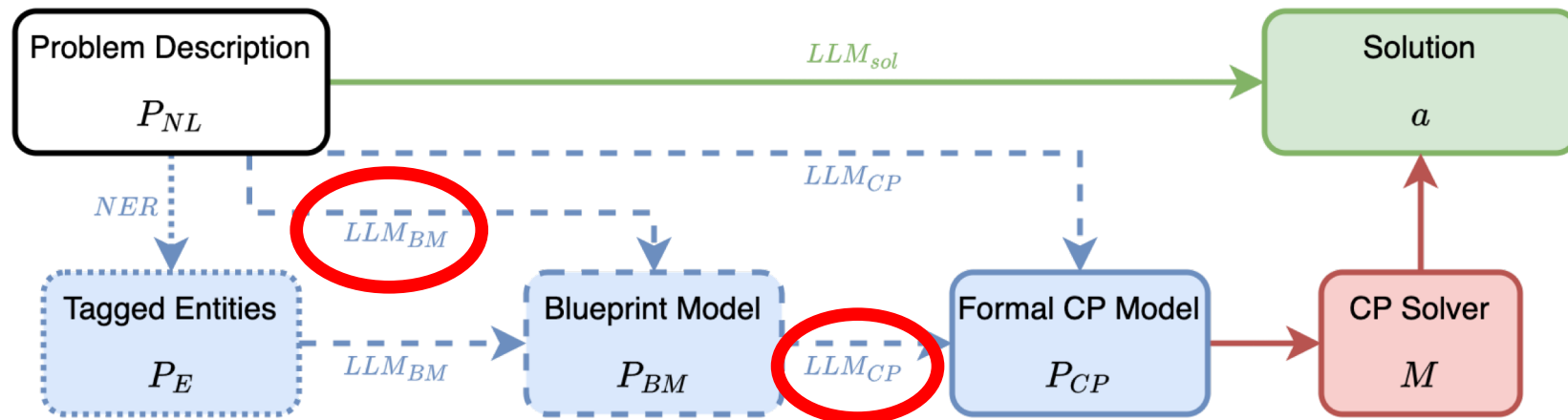
$$P_{CP} = LLM(C \oplus Q)$$

$$a = M(P_{CP})$$

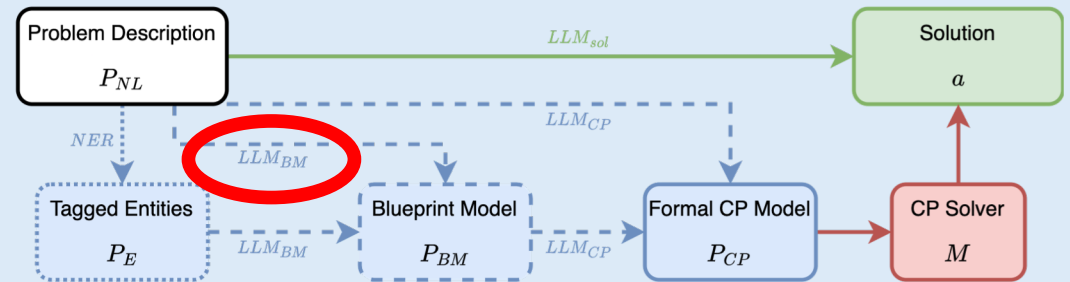
<i>j</i>	<i>Q_j</i> (Problem Desc)	<i>R_j</i> (CP Model)
1	A popular actor... Which movies to choose?	<pre>from cpmpy import *</pre> # Data movies = [{"title": "Tarjan of the Jungle", "interval": [4, 13]}, ...] # Constraints ... # Solve model.solve() 
...	...	...
<i>n</i>	...	...


 $\{(Q_j, R_j)\}_{j=1}^n$

3. Blueprint Model

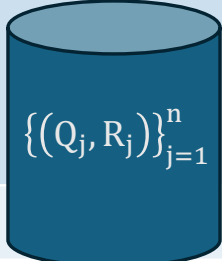


3. Blueprint Model

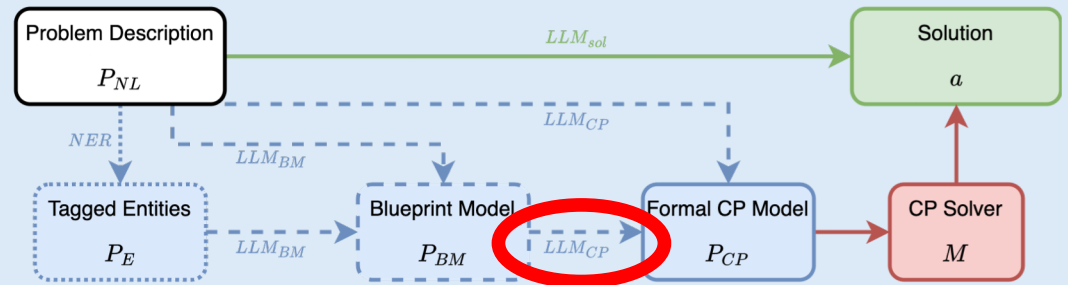


Decompose Model Generation:

$$P_{BM} = LLM(C \oplus Q)$$

j	Q_j (Problem Desc)	R_j (Blueprint Model)
1	A popular actor... Which movies to choose? 	# Decision Variables: Define binary variables to represent whether the movie is selected (1) or not (0). The variable 'selected_movies' is an array of size equal to the number of movies (9 in this case). # Domains: Each variable in 'selected_movies' has a domain of [0, 1] where 1 represents the movie is selected and 0 represents the movie is not selected. # Constraints: Non-overlapping: Ensure that the intervals of selected movies do not overlap. This can be represented by iterating over the movies and for each pair of selected movies (i, j), add a constraint that the intervals of the movies do not overlap. # Objective: Maximize the number of selected movies, which is equivalent to maximizing the sum of the selected_movies array.
...		...

3. Blueprint Model



Decompose Model
Generation:

$$P_{BM} = LLM(C \oplus Q)$$

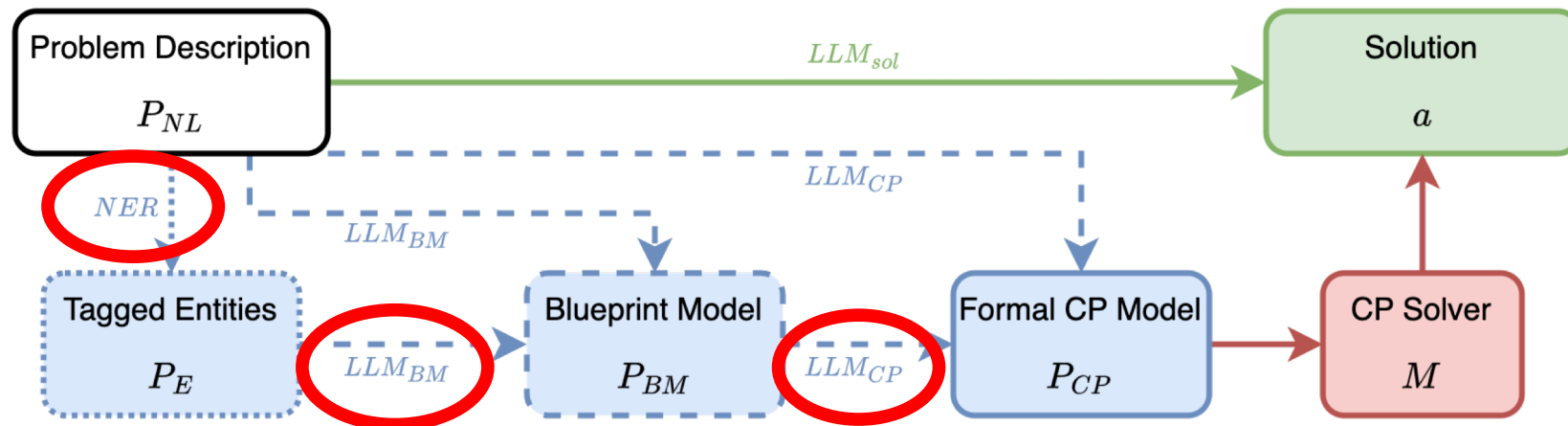
$$P_{CP} = LLM(C \oplus Q \oplus P_{BM})$$

$$a = M(P_{CP})$$

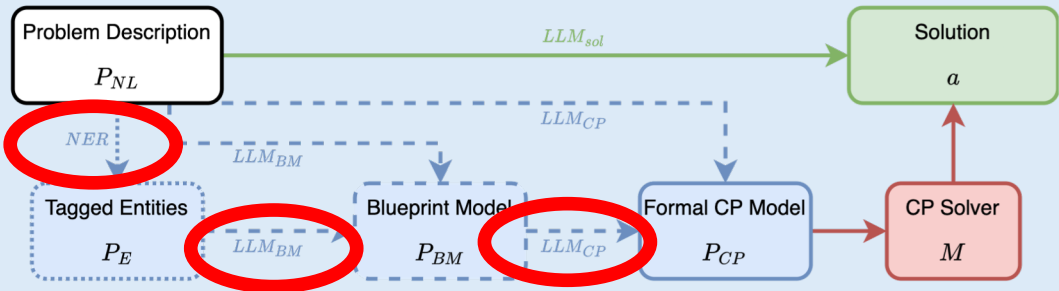
j	Q_j (Problem Desc $\oplus$ Blueprint Model)	R_j (CP Model)
1	A popular actor... Which movies to choose? # Decision Variables: Define binary variables to represent whether the movie is selected (1) or not (0). The variable 'selected_movies' is an array of size equal to the number of movies (9 in this case). ... # Objective: Maximize the number of selected movies, which is equivalent to maximizing the sum of the 'selected_movies' array.	<pre>from cpmPy import *</pre> # Data movies = [{"title": "arjan of the jungle", "interval": [4, 13]}, ...] # Constraints ... # Solve model.solve() </>
...	...	...

$\{(Q_j, R_j)\}_{j=1}^n$

4. Named Entity Recognition



4. Named Entity Recognition



j	Q _j (Problem Desc)	R _j (CP Model)
1	A popular actor... Which movies to choose? CONST (36-41): only one OBJ_DIR (44-49): maximize ... <Blueprint Model>	<pre>from cpmPy import * # Data movies = [{"title": "Tarjan of the Jungle", "interval": [4, 13]} ...] # Constraints ... # Solve model.solve()</pre>
...	...	...

$\{(Q_j, R_j)\}_{j=1}^n$

Evaluation Metrics

Declaration Accuracy

Model Accuracy

Solution Accuracy

#Errors: CP Model could not be run

```
15 # Decision Variables
16 selected_movies = boolvar(shape=num_movies) # 1 if the movie is selected, 0 otherwise
17
18 # CP Model
19 model = Model()
20
21 # Non-overlapping movie schedules
22 for i in range(num_movies):
23     for j in range(num_movies):
24         # Check if the intervals overlap for each pair of movies
25         if i != j and movies[i][2] > movies[j][1] and movies[j][2] > movies[i][1]:
26             # Constraint: the movies cannot be selected together
27             model += ~(selected_movies[i] & selected_movies[j])
28
29 # Objective: Maximize the number of selected movies
30 model.maximize(sum(selected_movies))
31
32 # Solve
33 model.solve()
34 print(selected_movies.value())
35
36 # Output:
37 # [False False False False False True True True True]
```

Experiments: Datasets

- **NL4Opt¹**: Linear Optimization Problems
- **LGP²**: Logic Grid Puzzles described with clues and entities
- **Mixed (new)**: 18 diverse problems drawn from a CP Modelling course.

¹ Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, et al. NI4opt competition: Formulating optimization problems based on their natural language descriptions.

² Arindam Mitra and Chitta Baral. Learning to automatically solve logic grid puzzles.

LLMs as CP solvers? No.
LLMs as CP modellers? Possible.

Dataset	Method	Accuracy (%)
NL4Opt	CoT Solving	11.46
	CP Modelling	81.31
LGPs	CoT Solving	9.36
	CP Modelling	57.00
Mixed	CoT Solving	16.67
	CP Modelling	50.00

Static 4-shot ICL, with gpt-3.5-turbo-0125

Intermediate Representations

1. Beneficial in small linear problems (NL4Opt)
2. For more complex problems mixed results, **noise** in the context:
 - NER: Misclassification of variables.
 - BM: Larger context, possible to generate errors.

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

Dataset	#Shots	CP	+ 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.

Retrieval-Augmented ICL? Yes.

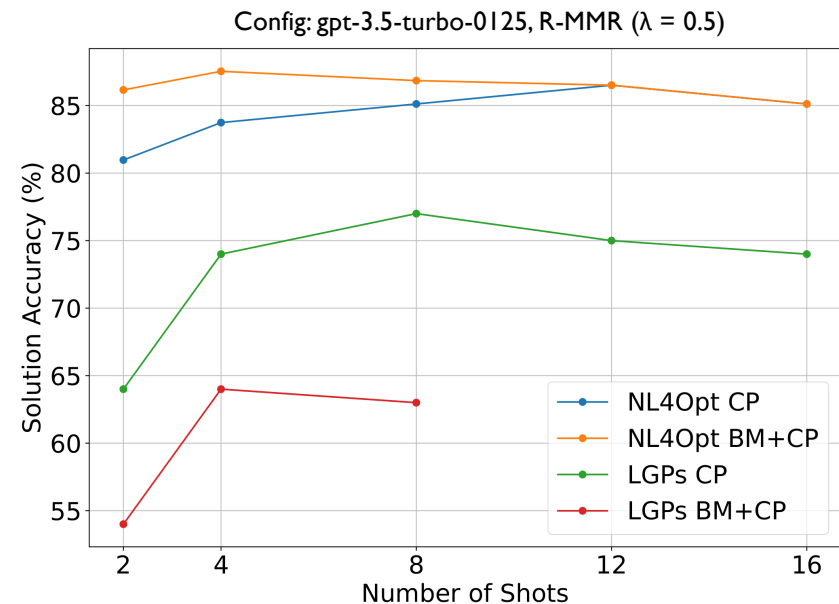
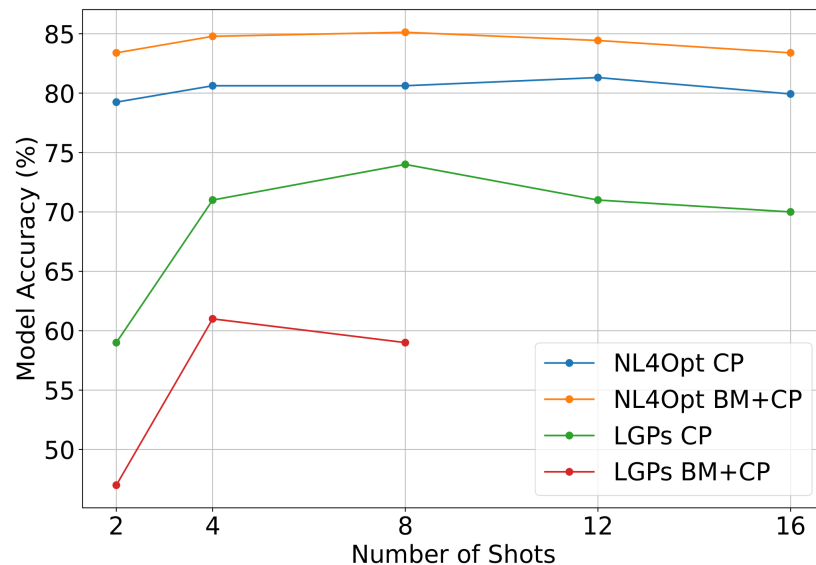
- Context **relevance** and **diversity** improve accuracy.
- Recency effect is influential.

Retrieval	LGP's Sol. (%)	LGP's Decl. (%)	NL4Opt Sol. (%)	NL4Opt Decl. (%)
Static	57.00	80.45	81.31	87.81
Random	66.00	76.75	77.16	85.60
Similarity	68.00	85.34	85.12	87.72
R-Similarity	72.00	89.83	83.39	88.16
Diversity	66.00	83.09	84.08	87.99
R-Diversity	74.00	87.98	83.74	87.10
Random w/ Last-Similar	74.00	89.91	83.74	88.34

CP model Config: gpt-3.5-turbo-0125, R-MMR 0.345.

The more examples the better?

Not exactly.



**Adding in-context examples improves
up to a threshold.**

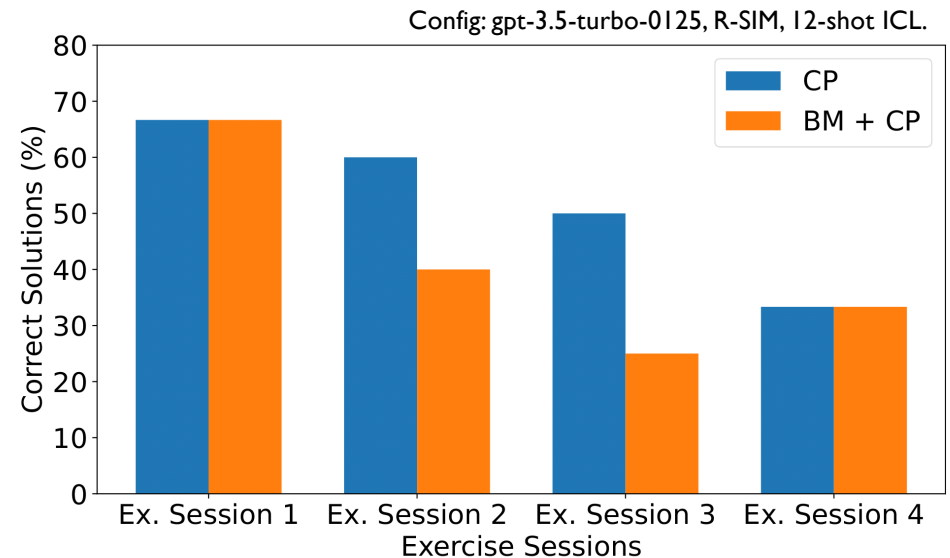
What about more diverse problems?

- Small Dataset: Few examples to retrieve from.
- More complex problems **harder** for LLMs as well.

- Evaluation challenges:

Problem: Choice of Decision Variables is free => ~~Model Accuracy~~.

LIGHTBULB Prompt LLMs to generate code that prints the solution in specific format => **Solution Accuracy**.



Key Conclusions



LLMs as CP solvers? No
LLMs as CP modellers? **Possible**



Retrieval-Augmented In-Context
Learning **improves** performance



Modelling more complex problems
still **challenging**

Future/Current Work

High-Quality NL-CP
dataset.

Beyond ICL:
Supervised **fine-tuning** of LLMs.

Integrate techniques
from LLMs **coding**
assistants.

Multi-turn
interactive
modelling system.

Demo

<https://chatopt.cs.kuleuven.be>

