Constraint Modelling with LLMs Using In-Context Learning

Kostis Michailidis
DTAI, KU Leuven, Belgium

Dimos Tsouros
DTAI, KU Leuven, Belgium

Tias Guns
DTAI, KU Leuven, Belgium

Abstract

Constraint Programming (CP) allows for the modelling and solving of a wide range of combinatorial problems. However, modelling such problems using constraints over decision variables still requires significant expertise, both in conceptual thinking and syntactic use of modelling languages. In this work, we explore the potential of using pre-trained Large Language Models (LLMs) as coding assistants, to transform textual problem descriptions into concrete and executable CP specifications. We present different transformation pipelines with explicit intermediate representations, and we investigate the potential benefit of various retrieval-augmented example selection strategies for in-context learning. We evaluate our approach on 2 datasets from the literature, namely NL4Opt (optimisation) and Logic Grid Puzzles (satisfaction), and a heterogeneous set of exercises from a CP course. The results show that pre-trained LLMs have promising potential for initialising the modelling process, with retrieval-augmented in-context learning significantly enhancing their modelling capabilities.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Computing methodologies → Natural language generation; Computing methodologies → Discrete space search

Keywords and phrases Constraint Modelling, Constraint Acquisition, Constraint Programming, Large Language Models, In-Context Learning, Natural Language Processing, Named Entity Recognition, Retrieval-Augmented Generation, Optimisation

Digital Object Identifier 10.4230/LIPIcs.CP.2024.20

Supplementary Material *Software (Source Code)*: `https://github.com/kostis-init/CP-LLMs-ICL` [37], archived at `swh:1:dir:5e4383ad6c4329796c9f21c51bbff4882dca8271`

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (Grant No. 101002802, CHAT-Opt).

Acknowledgements We want to thank the reviewers for their valuable feedback.

1 Introduction

Constraint Programming (CP) is a powerful paradigm for solving complex combinatorial decision-making and optimisation problems. It is widely applicable in various industrial tasks such as scheduling, resource allocation, and various assignment problems [49, 55]. However, utilizing CP requires the translation of real-world problems into a formal model; defining decision variables, constraints, and potentially an optimisation function. This is a complex process, necessitating expertise in the specific application domain, in addition to CP modelling and the semantic and syntactic formalisms of CP solvers or modelling languages.

The expertise needed for modelling CP problems has been recognised to be a bottleneck for the wider use of CP [16, 15]. This has motivated the development of tools and methods to simplify the modelling process, e.g. high-level modelling languages and automatic reformulation libraries such as MiniZinc [40] or CPMpy [21]. There is also active research on assisting the user through *constraint acquisition*, where either the user has to provide examples of solutions and optionally non-solutions (passive learning) [45, 35, 44] or the user interacts with the system, classifying partial assignments as solutions or not [3, 51, 50].

While these advancements show potential in assisting the user through the modelling process, there is still a gap between a natural language description of the problem and its corresponding CP model. As a result, a number of works have looked at utilizing Natural Language Processing (NLP) techniques in modelling. For example, in [27], a method to detect constraints from textual descriptions of combinatorial problems is presented, while [9] shows how Logic Grid Puzzle (LGP) clues can be formulated in first-order logic expressions. A classification-based approach to detect and rewrite the typical LGP clues as constraints was also explored [24]. Recently, the NL4Opt competition at NeurIPS[1] formulated the challenge of using NLP methods to transform textual descriptions of small linear programming optimisation problems into an LP formulation [47]. The most successful approaches of the competition [25, 22, 41, 18] primarily utilized fine-tuned BERT [12] and BART [29] architectures.

Orthogonally, the rise of new-generation Large Language Models (LLMs) pre-trained on web-scale data, such as GPT-3 [4] and similar models, has enabled the introduction of *coding assistants* [8]. A coding assistant is an LLM that can write a certain piece of code as a response to a user request in natural language. Given that these LLMs are trained on large portions of web data, they can likewise be asked to formulate constraint models for publicly documented constraint-solving systems. Indeed, there are preliminary reports on prompting strategies that present initial successes in CP modelling [52, 1]. On arithmetic and logical reasoning tasks, LLMs have also been used to rewrite the task into a formal specification that can be solved with an SMT Theorem Prover [62]. For LGPs, LLMs have been prompted to detect the constants and then rewrite complex clues and translate them to ASP rules [23]. On the NL4Opt dataset, pre-trained LLMs have been shown to outperform the fine-tuned BERT/BART models [43, 31].

These are promising precursors to employing LLMs to formulate CP models from natural language descriptions of any combinatorial problem, including logic grid puzzles, integer linear programs and more. While this is still a distant dream, it requires evaluating on both satisfaction (CSP) and optimisation problems (COP), for well-defined evaluation measures. We will define these and systematically investigate two orthogonal techniques that are commonly used in LLM approaches: 1) using multiple prompts and intermediate representations to get a final formal CP specification; and 2) using in-context examples and retrieval-augmented example selection techniques. Our goal is to contribute to the development and better understanding of coding assistants for CP modelling.

The contributions of this paper are as follows:

- We present a framework that transforms natural language descriptions of constraint problems into formal solvable CP models using LLMs.
- We compare prompting LLMs to generate solutions, to generating CP models that are then used to solve the problem at hand. We employ different approaches using intermediate representations, and various retrieval-augmented in-context examples selection techniques, including novel variations.

[1] `https://neurips.cc/virtual/2022/competition/50079`

- We define precise evaluation metrics, to standardize the evaluation of the performance of LLMs in modelling CP problems.
- We systematically evaluate the ability of pre-trained LLMs to generate accurate constraint models and solutions for satisfaction and optimisation problems, on 2 datasets from the literature and exercises from a CP course.

2 Background

This section formalizes some fundamental concepts needed for our methodology.

2.1 Constraint Programming

CP is a paradigm for modelling and solving combinatorial problems, by declaratively stating decision variables, their possible values (domains), the constraints that express the relationships between these variables, as well as potentially an objective function to optimise. We now formalize these elements for both satisfaction and optimisation problems.

Constraint Satisfaction Problems

A CSP can be formally defined as a tuple $(\mathbf{X}, \mathbf{D}, \mathbf{C})$, where:

- $\mathbf{X} = \{x_1, x_2, \ldots, x_n\}$ is a set of n decision variables.
- $\mathbf{D} = \{D_1, D_2, \ldots, D_n\}$ represents the domains of these variables, where each D_i is a subset of $\mathbb{Z}$ and specifies the allowable values for the decision variable x_i.
- $\mathbf{C} = \{C_1, C_2, \ldots, C_m\}$ is a set of m constraints, where each constraint C_j includes tuples of allowed values for a subset of the decision variables, formally $C_j \subseteq D_{j1} \times D_{j2} \times \ldots \times D_{jk}$ for some subset $\{x_{j1}, x_{j2}, \ldots, x_{jk}\} \subseteq \mathbf{X}$.

Let $sol(\mathbf{C})$ denote the set of all solutions to a CSP. An assignment $\mathbf{a} = \{x_1 = v_1, x_2 = v_2, \ldots, x_n = v_n\}$ is in $sol(\mathbf{C})$ if each $v_i \in D_i$ and if all constraints in $\mathbf{C}$ are satisfied, i.e., $(v_{j1}, v_{j2}, \ldots, v_{jk}) \in C_j$ for each $j \in \{1, 2, \ldots, m\}$.

Constraint Optimisation Problems

A COP extends a CSP by incorporating an objective function f that needs to be minimized or maximized. Formally, a COP is defined as a tuple $(\mathbf{X}, \mathbf{D}, \mathbf{C}, f)$, where $f : \prod_{i=1}^{n} D_i \to \mathbb{R}$ represents the objective function to be optimised. An optimal solution to a COP is an assignment $\mathbf{a}$, such that $\mathbf{a} \in sol(\mathbf{C})$, and $\mathbf{a}$ optimises the objective function f. For maximization, $f(\mathbf{a}) \geq f(\mathbf{b})$, for all $\mathbf{b} \in sol(\mathbf{C})$. For minimization, the inequality is reversed.

2.2 Large Language Models

Large Language Models (LLMs) are deep learning architectures with billions of parameters, based on the Transformer [54, 2]. They are extensively trained on diverse textual data, which allows them to learn and produce complex language patterns [30]. During inference, LLMs generate text by predicting the next token in a sequence, based on the previous tokens. Given an input text (or prompt) p, they produce a sequence of w tokens as follows:

$$LLM(p) = (x_1, x_2, \ldots, x_w), \quad \text{where } x_{t+1} = \arg\max_{x_{t+1}} P(x_{t+1}|p, x_{\leq t}) \tag{1}$$

Here, $x_{\leq t}$ denotes the sequence of tokens generated up to time step t, for all $t \in \{0, 1, ..., w-1\}$. $P(x_{t+1}|p, x_{\leq t})$ is the conditional probability of token x_{t+1} given the initial prompt p concatenated with the preceding tokens $x_{\leq t}$. Also, $x_{\leq 0}$ is the empty sequence and x_1 is the first generated token. The sequence generation process terminates when a predefined stop token is generated or when the maximum context length of the LLM is reached. In this work, we utilize greedy decoding for LLMs and assume the transformation of text into a token sequence and vice versa is done automatically.

In-Context Learning

In-Context Learning (ICL) received significant attention with the introduction of GPT-3 [4]. As a learning paradigm, it enables LLMs to adapt to new tasks at inference time by including examples directly in their input prompts [14, 28, 59]. This capability allows these models to generate responses that are contextually aligned without requiring retraining.

Formally, ICL specifies an ordered set E of k input-output pairs, denoted as $E = \{(i_j, o_j)\}_{j=1}^{k}$. During inference, the examples (or shots) in E are inserted before a new input to influence the model's output towards the expected task-oriented response. This can be represented as $LLM(E \oplus i_{new}) \approx o_{new}$, where o_{new} is the expected response for the input i_{new} based on the patterns outlined by the in-context examples E.

Retrieval Augmented ICL

A challenge in ICL is how to choose the in-context examples, with various studies on how to select [33, 61, 57, 65, 32], order [36, 34, 60], or formulate [58, 20, 26, 38] them. Some works used a predefined static set of examples [58], while others employed a more dynamic approach by selecting them from a database at inference time [63]. The latter, referred to as retrieval-augmented ICL (RAICL), aims to specifically adapt the context to each new input. To implement RAICL, we define a retrieval function R that selects the in-context examples E as follows:

$$E = R(i_{new}, S, k) \tag{2}$$

where i_{new} is a text input, and $S = \{(i_j, o_j)\}_{j=1}^{n}$ is the database of examples that the retrieval function will choose from, with $n \geq k$ and thus $E \subseteq S$. Various strategies – i.e. implementations of the retrieval function R – can be utilised for dynamically selecting in-context examples. They are typically relevance-based metrics with respect to i_{new}. We describe some in 3.4.

3 Methodology

We now formalize the framework of our study; a baseline solution generation method, the proposed modular pipeline to produce an executable CP model, and the RAICL strategies.

3.1 Problem Formulation

The objective is to create a system that solves constraint problems given a natural language description P_{NL}. More formally:

Input: A natural language description of a constraint problem P_{NL}, from which decision variables, constraints, and potentially an objective function must be inferred.

Output: A valid solution or assignment $\mathbf{a}$ of values to the decision variables, satisfying all constraints and optimising the objective function if specified, as described in 2.1.

The methods to derive **a** from P_{NL} will be explored through different approaches. Firstly, we will consider the LLMs as constraint problem solvers, where they generate reasoning and ultimately the solution. More fundamentally, we will prompt LLMs to produce solvable CP models, through optional intermediate representations, and use a solver to compute the solution. The steps and pathways that we will explore are visible in Figure 1.

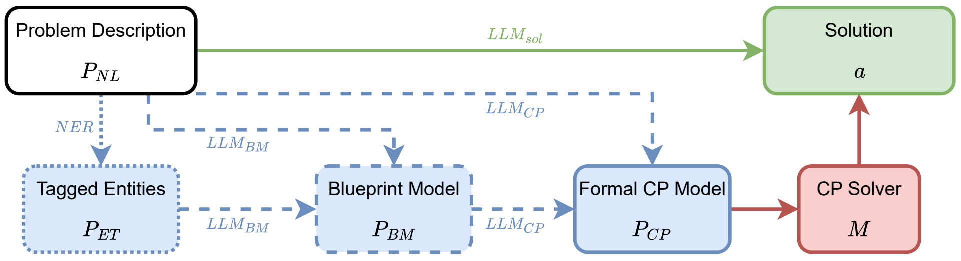


Figure 1 The various pathways for solving constraint problems: direct LLM-based solving (LLM_{sol}), or solvable model generation (LLM_{CP}), optionally including entity tagging (NER) and blueprints (LLM_{BM}) as intermediate representations.

3.2 LLMs as CP Solvers

As a baseline method, we investigate the use of LLMs to directly generate the solution from the natural language problem description P_{NL} of a constraint problem. This approach is based on Chain-of-Thought (CoT) techniques, where the model is prompted to produce a sequence of reasoning steps towards solving a given arithmetic or symbolic reasoning problem [58]. The solution generation process is formalized as follows:

$$\hat{\mathbf{a}} = LLM(E \oplus P_{NL}), \quad \text{where } E = R(P_{NL}, S_{NL-CoT}, k) \tag{3}$$

Here, LLM represents the large language model as defined in (1), E is the set of in-context examples retrieved from the full examples database S_{NL-CoT}, and R is the retrieval function (2). In S_{NL-CoT}, each i_j is a natural language description of a constraint problem and o_j is its CoT reasoning that ends with the solution. The input to the LLM is the concatenation of the in-context examples E with the textual description of the problem P_{NL}, and the output is the generated reasoning sequence $\hat{\mathbf{a}}$. The final solution **a** is then extracted from the end of this generated sequence.

3.3 LLMs as CP Modellers

Recognizing the limitations of LLMs in performing arithmetic and logical computations [17], some recent works shifted towards using them to generate formal models or programs [19, 52, 23, 1]. Inspired by these studies, we introduce an additional step in which the LLM is tasked with generating a formal CP model P_{CP} from the given constraint problem description P_{NL}. Our methodology requires that P_{CP} is formatted to fit a predefined CP solver or modelling framework. We define a two-step model-and-solve process as follows:

1. **Model Generation:**

$$P_{CP} = LLM(E \oplus P_{NL}), \quad \text{where } E = R(P_{NL}, S_{NL-CP}, k) \tag{4}$$

The only difference with Equation 3 is the content of the database S_{NL-CP}. Here, each i_j is a constraint problem description in natural language and o_j is its corresponding formal CP model, formatted according to the predefined CP solver or modelling framework. An example of a CP model can be seen in Figure 2, bottom right.

2. **Model Solution:**

$$\mathbf{a} = M(P_{CP}) \tag{5}$$

M represents the predefined off-the-shelf CP solver or modelling framework that computes the solution $\mathbf{a}$ from the generated formal model P_{CP}.

3.3.1 Blueprint Model Generation

Transforming textual descriptions directly into formal runnable CP models presents a complex challenge. It could be seen as involving two separate non-trivial operations: identifying elements of the constraint problem and generating solver-compatible output. We propose to use different LLM calls and examples for each of the two. We refer to the intermediary result as the blueprint model (P_{BM}). It outlines in plain text the decision variables, constraints and objective function (if applicable) based on the problem description, each of them in both natural language and mathematical notation. An example is provided in Figure 2, bottom left.

We first prompt an LLM to transform the problem description P_{NL} into a blueprint model P_{BM}. Then, we use the description combined with the blueprint to generate the formal CP model. This approach aims to decompose the complex task of direct model generation and also provides an additional interpretable layer. We define this pipeline as follows:

1. **Blueprint Generation:**

$$P_{BM} = LLM(E \oplus P_{NL}), \quad \text{where } E = R(P_{NL}, S_{NL-BM}, k) \tag{6}$$

In S_{NL-BM}, each i_j is a constraint problem description in natural language and o_j is its corresponding blueprint model.

2. **Model Generation:**

$$P_{CP} = LLM(E \oplus P_{NL} \oplus P_{BM}), \quad \text{where } E = R(P_{NL} \oplus P_{BM}, S_{NL-BM-CP}, k) \tag{7}$$

In $S_{NL-BM-CP}$, each i_j is the concatenation of a constraint problem description with its blueprint model and o_j is their corresponding formal CP model.

3. **Model Solution:** Same as (5).

3.3.2 Named Entity Recognition

As part of the first subtask of the NL4Opt competition, there have been numerous approaches for accurately tagging and identifying linear optimisation entities from textual descriptions [56, 41, 13]. Based on these works, we integrate Named Entity Recognition (NER) into our methodology to systematically identify and extract decision variables, parameters, constraints, and objective keywords from natural language descriptions P_{NL} of constraint problems. Parameters are fixed coefficients or constants in the formal definition of the CP model. For example, in Figure 2 the numeric values in the description are parameters of the problem.

This integration of entity tags aims to improve the available information for constructing blueprints and formal CP models as follows:

1. **Entity Tagging:** $P_{ET} = NER(P_{NL})$, where NER is an automated system designed to detect and label specific entities relevant to CP problems within the text of P_{NL}. This step could also be performed with an LLM, but we chose NER4Opt as a specialized framework trained for entity tagging in an optimisation context [11].

A retired professor wants to invest up to \$50000 in the airline and railway industries. Each dollar invested in the airline industry yields a \$0.30 profit and each dollar invested in the railway industry yields a \$0.10 profit. A minimum of \$10000 must be invested in the railway industry and at least 25% of all money invested must be in the airline industry. How to maximize the professor's profit?

Decision Variables:
- Amount invested in the airline industry: *Airline*
- Amount invested in the railway industry: *Railway*

Constraints:
- Total investment should not exceed 50000 dollars: $\mathit{Airline} + \mathit{Railway} <= 50000$
- Minimum investment of 10000 dollars in the railway industry: $\mathit{Railway} >= 10000$
- At least 25% of all money invested must be in the airline industry: $\mathit{Airline} >= 0.25 \times (\mathit{Airline} + \mathit{Railway})$

Objective:
- Maximize profit (\$0.30 profit per dollar invested in the airline, \$0.10 profit per dollar invested in the railway industry): $0.30 \times \mathit{Airline} + 0.10 \times \mathit{Railway}$

```python
from cpmpy import Model, intvar

# Decision Variables
Airline = intvar(0, 1_000_000)
Railway = intvar(0, 1_000_000)

# Constraints
m = Model()

# Total investment
m += Airline + Railway <= 50000
# Minimum investment in railway
m += Railway >= 10000
# Minimum investment in airline
m += Airline >= 0.25 * (Airline + Railway)

# Objective: Maximize profit
m.maximize(0.3 * Airline + 0.1 * Railway)

m.solve()
```

Figure 2 An example of textual description (top), blueprint (bottom left), and formal CP model written with the CPMpy library (bottom right) for the investment problem from NL4Opt.

2. **Blueprint Generation:**

$$P_{BM} = LLM(E \oplus P_{NL} \oplus P_{ET}), \quad \text{where } E = R(P_{NL} \oplus P_{ET}, S_{NL-ET-BM}, k) \tag{8}$$

   In $S_{NL-ET-BM}$, each i_j is the concatenation of a constraint problem description with a textual representation of its tagged entities and o_j their corresponding blueprint model.
3. **Model Generation:**

$$\begin{aligned} &P_{CP} = LLM(E \oplus P_{NL} \oplus P_{ET} \oplus P_{BM}), \\ &\text{where } E = R(P_{NL} \oplus P_{ET} \oplus P_{BM}, S_{NL-ET-BM-CP}, k) \end{aligned} \tag{9}$$

   Here, in $S_{NL-ET-BM-CP}$ each i_j is a constraint problem description concatenated with its tagged entities and blueprint model, and o_j is its corresponding CP model.
4. **Model Solution:** Same as (5).

3.4 In-Context Examples Selection

A significant component of the presented pipelines is the retrieval function R, as it is responsible for the dynamic selection of in-context examples from the full database S (2.2).

Static & Random Strategies

As a baseline method, we can employ a traditional approach where the same predefined examples are selected for any input [58]. These examples are selected in advance and remain unchanged, providing a consistent basis for evaluation. In our research, we retrieve the first k example pairs from S, formalized as:

$$R(i_{new}, S, k) = \{S_j \mid j \in \{1, 2, \dots, k\}\} \tag{10}$$

where S_j represents the j-th example tuple in the database.

To provide a more stochastic view, we will also consider random selection where we use the same selection strategy but on a randomly shuffled database.

Semantic Similarity (SIM)

The first retrieval-augmented strategy that we will utilize selects examples that are semantically close to the input [6]. As it provides context that is relevant to the current query, it has been shown to improve ICL on various tasks [33]. We define semantic similarity Sim between two texts using the cosine similarity between their vector embeddings:

$$Sim(\text{text}_1, \text{text}_2) = \frac{\vec{v}(\text{text}_1) \cdot \vec{v}(\text{text}_2)}{\|\vec{v}(\text{text}_1)\| \|\vec{v}(\text{text}_2)\|} \tag{11}$$

where $\vec{v}(\text{text})$ is the vector embedding of a text. Vector embeddings are numerical representations of tokens that capture semantic meanings, often derived from the embedding layers of LLMs. For a new input i_{new}, we first compute its embedding vector and then the retrieval function R selects the k most semantically similar examples from S as follows:

$$R(i_{\text{new}}, S, k) = \{(Sorted(i_{\text{new}}, S))_j \mid j \in \{1, 2, \dots, k\}\}, \tag{12}$$

$$Sorted(i_{\text{new}}, S) = \{S_j \mid Sim(i_{new}, i_j) \geq Sim(i_{new}, i_{j+1}) \quad \forall j \in \{1, 2, \dots, n-1\}\}, \tag{13}$$

where i_j is the input element of the j-th example pair (i_j, o_j) in S (2.2).

Maximal Marginal Relevance (MMR)

To add more information and variety in the context, we will also consider the MMR metric [5, 63]. It selects examples that balance relevance to the input with diversity within the chosen set. This balance is achieved by selecting an example S_j that maximizes both its relevance to the input and its difference from previously selected examples:

$$\arg\max_{S_j \in S \setminus T} \left[\lambda \cdot Sim(i_{new}, i_j) - (1 - \lambda) \cdot \max_{S_t \in T} Sim(i_j, i_t) \right] \tag{14}$$

Here, λ is the hyperparameter that controls diversity, Sim is the similarity measure (11), S is the total set of examples, and T is the set of already selected examples. The retrieval function R will first select the most similar example from the database S, and select the remaining $k - 1$ according to (14).

Last-Similar Variations

We also introduce variations of example selection strategies, inspired by the recency effect [10, 53, 64]. This suggests that the content positioned towards the end of an input sequence has a more significant influence on the output of an LLM [34]. Based on this insight, the next proposed strategies involve placing the most semantically similar example last in the sequence to exploit this recency bias.

- **Reversed Semantic Similarity (R-SIM)**: The retrieved examples based on the Semantic Similarity metric are reordered so that the most relevant to the current problem is last. This was also explored in Liu et al. [33].
- **Reversed Maximal Marginal Relevance (R-MMR)**: Similarly with R-SIM, the original MMR order is reversed, so that the most similar example is placed last.
- **Last-Similar, Rest-Random (LSRR)**: Places the most similar example from S last, and the rest are selected randomly.

4 Experiments

This section outlines our experimental framework, devised to evaluate the effectiveness of using LLMs to convert natural language descriptions into formal CP models. We seek to answer the following research questions:

Q1: How does each intermediate representation impact the efficiency of LLMs in generating CP models, and how do they compare to direct solution generation from the LLMs?

Q2: How do different in-context example selection strategies influence the correctness of the generated CP models and their solutions?

Q3: How many in-context examples should be used, depending on the type of the problems?

Q4: How effectively can LLMs with RAICL generate CP models for a small dataset of problems that human students learn to solve?

4.1 Setup & Datasets

For our experimental setup, we utilized Python 3.9 along with several specialized tools and libraries[2]. For entity tagging, we employed NER4Opt [11] and for CP modelling we used CPMpy [21]. To implement and test RAICL strategies, we used LangChain [7] and Chroma DB[3], and the OpenAI API[4] was used for accessing gpt-3.5-turbo-0125 and vector embeddings. Our framework is evaluated on the following datasets:

- **NL4Opt [47]:** This dataset includes NL descriptions of linear optimisation problems, with 289 test and 713 training instances, such as in Figure 2. We use the test instances for evaluation, and the training instances to compose the examples database S.
- **Logic Grid Puzzles (LGPs) [39]:** Consists of 50 train and 100 test instances featuring logical puzzles described with clues and entities; these can be expressed as CSPs. As above, we use the test instances for evaluation and the training instances for S.
- **Mixed CP Dataset:** Comprises 18 diverse CP problems (a mix of 13 CSPs and 5 COPs) drawn from a university-level CP modelling course, arranged by increasing complexity. Due to its small size, we use a leave-one-out strategy for evaluation, testing each problem individually while utilizing the rest for S.

To facilitate both RAICL and evaluation, we extended the datasets by generating entity tags, blueprint models and formal CPMpy models. This generation process is described in Section 4.2. For the mixed CP dataset, we manually curated and assessed all the blueprint and formal CP models. For more information regarding these datasets, such as the average number of decision variables, constraints, and more, refer to Appendix E.

4.2 Data Annotations Generation

For NL4Opt and LGPs, we utilized NER4Opt [11] to generate the entity tags for all instances (3.3.2). Then, we created detailed CoT-including solutions for 4 instances to evaluate the baseline of our methodology (3.2). Finally, we manually created the blueprint and formal CP models of their first six instances, ensuring clarity and correctness. For generating the remaining ground-truth blueprint and CP models, we employed gpt-4-0125-preview [42] with static ICL using the manually produced ones. We ensured the correctness of the generated

[2] The code is available at `https://github.com/kostis-init/CP-LLMs-ICL`.
[3] `https://docs.trychroma.com/`
[4] `https://platform.openai.com/docs/api-reference`

CP models – correcting them when needed – as follows: For NL4Opt, we asserted that the generated models are equivalent (18) and produce the same solutions with the already existing canonical formulations. This is a deterministic procedure, as the linear constraints can be automatically transformed into a CP model. For LGPs, we validated that the solutions produced by the generated models match the ground-truth solutions already present in the dataset.

4.3 Evaluation

Defining a unified evaluation framework for modelling varying CP problem types from textual descriptions is not trivial. Evaluations at either the solution, constraint, or model level must include some form of mapping between the decision variables (and/or their values) of the predictions and those of the ground truth. We implemented a process that identifies the best match for each decision variable by first considering exact matches, then prefixes, substrings, and finally, a composite textual and numerical similarity metric.

Additionally, LLMs generate executable CP models that may contain syntax errors[5]. Therefore, we also track and report the number of models containing at least one error, denoted as **#Err** in the results. To ensure precise evaluation, we adopt a strict criterion: if a generated CP model cannot be executed due to syntax errors, or if the decision variables of the predicted model cannot all be mapped to those of the ground truth models, then the instance is considered incorrect on all metrics.

We now present three separate accuracy measures, each focused on different aspects.

Solution Accuracy

The solution accuracy metric evaluates the correctness of the solutions produced by the generated CP models compared to the ground truth solutions. A solution is deemed correct if it satisfies all constraints and, if applicable, achieves the optimal objective value as defined in the ground truth CP model. We formalize the solution accuracy as follows:

$$acc_{sol} = \frac{\sum_{i=1}^{N} valid(\mathbf{a}_i, true_i)}{N} \tag{15}$$

where N is the total number of test instances, $\mathbf{a}_i$ represents the solution derived from the LLM or the predicted CP model $pred_i$; $valid(\mathbf{a}_i, true_i)$ equals 1 if $\mathbf{a}_i \in sol(\mathbf{C}_{true_i})$, and for COPs $f_{true_i}(\mathbf{a}_i)$ must be optimal. We do not compute all solutions, but simply check whether $\mathbf{a}_i$ is a satisfying solution and optionally whether its objective value equals the optimal value.

Declaration Accuracy

As part of the evaluation metrics used in the NL4Opt competition [47], declaration accuracy measures the percentage of individual declarations (constraints and objectives) predicted accurately with respect to the ground truth declarations. It is calculated using the formula:

$$acc_{decl} = 1 - \frac{\sum_{i=1}^{N} \min(\mathrm{FP}_i + \delta_i, Q_i)}{\sum_{i=1}^{N} Q_i} \tag{16}$$

where N is the total number of test instances and Q_i is the total number of declarations, in the i-th ground-truth model. FP_i denotes the number of false positives, and δ is the difference in the number of constraints between the ground-truth model and the predicted model, only counted when the predicted model has fewer constraints.

[5] An example is demonstrated in Appendix C.2

To calculate FP_i, we count all the declarations of the predicted model for which an *equivalent* declaration was not found in the ground-truth model. Two constraints from different models are considered *equivalent* if they imply each other for the selected decision variable mapping. For verifying the correctness of a linear objective function, equivalence is established by asserting that the coefficients of the mapped variables are identical.

Model Accuracy

Model accuracy assesses the semantic correctness of the generated CP models relative to the ground truth models. It quantifies how well the entire set of constraints and the logical structure in a predicted model capture the problem as defined in the ground truth. This metric is defined as:

$$acc_{model} = \frac{\sum_{i=1}^{N} equiv(pred_i, true_i)}{N} \tag{17}$$

where $equiv(pred_i, true_i)$ equals 1 if the predicted model $pred_i$ is logically equivalent to the ground truth model $true_i$ for a specified mapping of variables. Logical equivalence is confirmed if the constraints of the predicted model imply and are implied by the constraints of the ground truth model, ensuring a bidirectional logical consistency:

$$equiv(pred, true) \Leftrightarrow ((\bigwedge pred \implies \bigwedge true) \wedge (\bigwedge true \implies \bigwedge pred)) \tag{18}$$

This verifies that the complete set of constraints in one model logically corresponds to those in the other, confirming their semantic equivalence. The equivalence algorithm is described in Appendix B.

4.4 Results

This section details the results and findings from our experiments with the proposed methods and systematically explores the proposed research questions. We do not present any 0-shot attempts, primarily due to the infeasibility of producing runnable CPMpy models without examples in the prompt context[6]. For the core experiments, we employed gpt-3.5-turbo-0125 with a temperature value of 0. Additional results on other LLMs are available in Appendix A.

4.4.1 Intermediate Representations

An incremental analysis is necessary to understand the effect of each intermediate component in our pipeline. As an initial baseline, we employ static (3.4) in-context examples selection, with $k = 4$ examples in the context. The results are shown in Table 1.

Using LLMs as CP modellers, instead of direct combinatorial problem-solvers, proved to be essential for obtaining substantial performance across the datasets. Even if the LLM is instructed to solve the problem with 4 pairs of descriptions and CoT solutions in the context, the direct solving approach results in significantly lower solution accuracy.

Interestingly, appending NER in the pipeline of the LGP instances nearly doubled the number of errors and dropped accuracy compared to direct CP modelling. Our qualitative analysis showed that decision variables were often misclassified as parameters, leading to inconsistencies and irrelevant context at subsequent steps in the pipeline [48]. Conversely, in the NL4Opt instances, where NER tagging was more accurate, the performance improvements were comparable to those achieved with BM. The inclusion of BM was slightly more beneficial

[6] An example with errors is demonstrated in Appendix C.1

Table 1 Comparison of our methods for both NL4Opt (#289 test instances) and LGPs (#100 test instances). Configuration: gpt-3.5-turbo-0125, 4-shot static in-context examples selection.

Dataset	Method	#Err	Accuracy (%)		
			Solution	Declaration	Model
NL4Opt	Direct	-	11.46	-	-
	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	Direct	-	9.36	-	-
	CP	**11**	57.00	**80.45**	55.00
	+ BM	18	**58.00**	70.69	**58.00**
	+ NER	20	54.00	67.77	50.00

than directly generating CP models, but it produced a larger number of errors in the LGP dataset. This can be attributed to the larger structure of the intermediate models in LGPs[7]. As such, LGP blueprints are prone to be generated with errors, which would introduce irrelevant context in the LLMs, similar to NER.

Comparing the results of the two datasets, we can see a discrepancy that can be attributed to their different nature and modelling difficulties. On the one hand, NL4Opt contains simple linear optimisation problems with around 2 decision variables and 3 constraints on average per instance. LGPs, in contrast, include 12 decision variables and 4 clues on average, including complex constraints such as *all different* and *pairwise exclusive disjunction*.

Overall, the best approach for both datasets is the blueprint and CP model generation, thus we will utilize this to investigate RAICL in the following experiments.

4.4.2 Examples Selection Strategy

To understand the impact of RAICL and different in-context example selection strategies, we evaluated the presented methods with a fixed number of $k = 4$ in-context examples. The results are shown in Table 2.

In both datasets, the dynamic retrieval algorithms consistently outperform static and random selection methods, underlining the importance of semantic relevance in the prompt context.

As proposed, applying the recency effect by placing the most similar example last in the context considerably improves accuracy. This is mostly evident in the large accuracy difference between the random and LSRR methods across both datasets; and additionally between the reversed retrieval methods in the LGPs with direct CP modelling. In this configuration, LSRR also outperforms MMR and R-MMR, which suggests that adding even more diversity further improves performance in LGPs.

In the NL4Opt dataset, the dynamic methods showed relatively similar performance, with all of them achieving higher accuracies than the static and random strategies. Declaration accuracy in this dataset is on par with the LP-specialized top-ranked approaches in the NL4Opt competition[8] [18]. In LGPs, employing retrieval-augmented ICL not only improved accuracy over all metrics but also significantly diminished the number of errors. This showcases that relevance in the context is important for generating valid and runnable code.

[7] An example is available in Appendix F
[8] `https://nl4opt.github.io/`

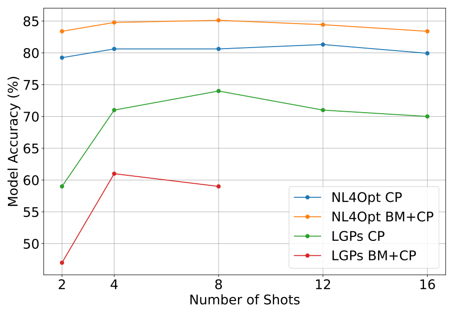
Table 2 Comparison of example selection strategies for NL4Opt and LGPs datasets, gpt-3.5-turbo-0125, 4-shot ICL, $\lambda = 0.5$.

Dataset	Method	CP				BM + CP			
		#Err	Accuracy (%)			#Err	Accuracy (%)		
			Sol.	Decl.	Mod.		Sol.	Decl.	Mod.
NL4Opt	Static	7	81.31	87.81	79.24	8	84.43	89.93	82.01
	Random	11	77.16	85.60	75.09	14	78.55	83.13	77.16
	SIM	8	**85.12**	87.72	80.28	**5**	86.16	90.02	84.08
	R-SIM	11	83.39	88.16	80.62	11	85.47	89.31	83.04
	MMR	8	84.08	87.99	**80.97**	7	86.51	**90.81**	**85.47**
	R-MMR	**6**	83.74	87.10	80.62	8	**87.54**	89.93	84.78
	LSRR	12	83.74	**88.34**	80.62	11	83.39	88.25	82.70
LGPs	Static	11	57.00	80.45	55.00	18	58.00	70.69	58.00
	Random	14	66.00	76.75	62.00	8	59.00	81.51	52.00
	SIM	9	68.00	85.34	66.00	9	66.00	81.24	63.00
	R-SIM	**4**	72.00	89.83	69.00	19	61.00	73.18	58.00
	MMR	10	66.00	83.09	64.00	7	63.00	83.75	58.00
	R-MMR	6	74.00	87.98	71.00	7	64.00	81.77	61.00
	LSRR	**4**	**76.00**	**89.96**	**72.00**	**5**	**75.00**	**86.39**	**70.00**

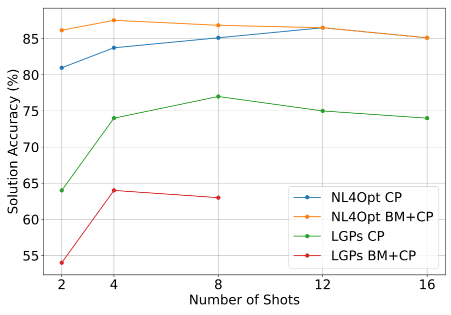
The overall results indicate that RAICL outperforms static selection in generating formal CP models by a large margin for both datasets. Additionally, the results of R-MMR and LSRR highlight the importance of both the last in-context example and the context diversity in the prompt.

4.4.3 Number of In-Context Examples

The results shown in Figure 3, illustrate how the number of in-context examples influences the model and solution accuracy in NL4Opt and LGPs, both for including blueprint model generation and not. We selected R-MMR as a retrieval strategy as it demonstrated consistently fewer errors and high accuracy across the two datasets.



(a) Model accuracy changes with the number of shots.



(b) Solution accuracy changes with the number of shots.

Figure 3 Comparison of model and solution accuracies with a varying number of in-context examples. Config: gpt-3.5-turbo-0125 R-MMR ($\lambda = 0.5$).

In the NL4Opt dataset, we observe a progressive gain in accuracy metrics as the number of in-context examples increases, reaching an optimal performance at 12 shots when not including BM. However, beyond this point, a further increase shows a slight performance degradation. Including the blueprint model caused this decline to start earlier, at 4–8 shots, hinting towards a ceiling effect on the size of the prompt context. The LGPs dataset presents a similar picture but with the BM inclusion consistently underperforming in comparison to the direct CP model generation. As in NL4Opt, including blueprint models peaks at a smaller k value (4) than when directly generating the CP model (peaking at $k = 8$). It should be noted that with blueprint models, the examples that generate the CP model are much larger as they also contain these blueprints.

Based on these observations, a balance is needed between providing adequate context and avoiding information overload, which can influence the quality of the provided answers. The number of in-context examples should also be adapted according to their size so that the prompt does not become overly dense. Notably, the LGPs with the blueprint model generation could not accommodate more than 8 in-context examples due to their length and the limitations imposed by the 16k context window of gpt-3.5-turbo-0125.

4.4.4 Mixed CP Dataset

We complete this section by assessing the potential of LLMs to generate correct CP models from a dataset with diverse and more complex constraint problems. We focused exclusively on solution accuracy since there can be multiple different CP models for such complex problems, which may involve varying selections of decision and auxiliary variables. As a result, the variable mapping required to use the other metrics proved impractical.

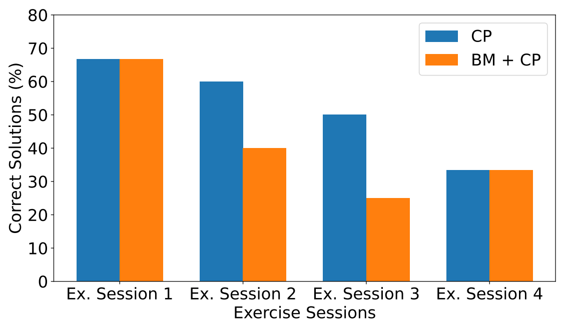
For these problems, we included additional instructions in the problem descriptions, specifying how solutions should be printed and formatted by the generated CP model code. This allowed us to directly validate this output with the ground truth model, ensuring that the solutions can be assessed without needing to map decision variables. Also, we chose R-SIM as the selected retrieval-augmented technique to balance the diversity of the dataset. The results are shown in Table 3, with supplementary results in Appendix D.

While adding more in-context examples improves the accuracy of the produced solutions, blueprint inclusion degrades performance. This suggests that its integration might be less beneficial for complex problems, in which the generated blueprints are even more likely to contain errors. Additionally, using retrieval-augmented R-SIM does not seem to improve much over static selection. Due to the small dataset size, static and R-SIM often share some of the same examples. This limited size and high diversity of the dataset proves challenging for LLMs, with only a bit over half of the exercises solved.

Table 3 Solution Accuracy across different numbers of in-context examples for both static and R-SIM retrieval strategies in the mixed CP dataset. LLM: gpt-3.5-turbo-0125.

#Shots	**Solution Accuracy (%)**			
	Static		**R-SIM**	
	CP	**BM + CP**	**CP**	**BM + CP**
2	33.33	16.67	50.00	38.89
4	50.00	50.00	55.56	38.89
8	50.00	44.44	55.56	44.44
12	55.56	50.00	61.11	44.44
16	50.00	-	61.11	-

As these problems are based on a university course, we arranged them in increasing difficulty based on the exercise session they were covered in. Figure 4 shows that as the difficulty of the problems increases, there is also an analogous decrease in accuracy, pointing to the challenge LLMs face when required to produce CP models for complex problems.



Exercise sessions:

1. 6 problems: Five Floors, Bank Card, Guards and Apples, Magic Square, Thick as Thieves, Money Change.
2. 5 problems: Color Simple, Movie Scheduling, Subset Sum, Subsets 100, Maximal Independent Sets.
3. 4 problems: Exodus, People in a Room, Kidney Exchange, Farmer and Cows.
4. 3 problems: Grocery, Climbing Stairs, Hardy 1729.

Figure 4 Solution accuracy per session when increasing difficulty. Config: R-SIM, 12-shot ICL.

5 Conclusion & Future Work

In this paper, we explored the potential of leveraging pre-trained Large Language Models to model CP problems from textual problem descriptions. We introduced and systematically evaluated LLM-based approaches, investigating the use of intermediate representations with multiple prompting steps, to formulate an executable CP model specification. We utilized in-context learning and retrieval techniques for the in-context examples. As the use of LLMs for modelling constraint problems has not been explored in much depth in the literature, we also focused on both augmenting existing datasets and defining evaluation measures. We augmented the NL4Opt and LGP datasets, with intermediate representations and formal model specifications in CPMpy, and created a small course-based diverse CP dataset. We presented an evaluation framework with precise metrics for the correctness of solutions, the semantic equivalence of the individual constraints, as well as the overall constraint model.

Our experiments demonstrated how challenging it is for LLMs to solve combinatorial problems directly. However, using them to write constraint models was significantly more successful. Including NER, as proposed in the NL4OPT challenge, did not always improve the quality of the final models. On the other hand, using a human-interpretable blueprint modelling before the executable model generation was beneficial only for the simple linear optimisation problems of NL4Opt. We observed that such intermediate steps can introduce additional errors, due to incorrect tagging and irrelevant prompt context. What mostly improved the performance of LLMs is: a) retrieval-augmented selection of the in-context examples, based on a balance of relevance to the current input and context diversity, and b) increasing the number of such examples up to a certain threshold per dataset. We also demonstrated that the small mixed CP dataset was a harder challenge, because of both the complexity of the problems and the small amount of available CP problems to effectively utilize RAICL, highlighting the need for creating more such datasets.

Further advancements towards the development and evaluation of CP modelling assistants require more high-quality NL-CP datasets, including pairs of problem descriptions and their formal CP model, potentially inspired by educational material or examples from modelling systems. This could also allow going beyond in-context learning and towards supervised fine-tuning of LLMs. In addition, addressing larger (parameterized) problems that include external data files is an interesting challenge. Finally, as the generated models will likely have occasional errors, the integration with other (code-specialized) LLMs, debugging techniques, or example-based constraint acquisition techniques holds much promise.

References

1 Boris Almonacid. Towards an automatic optimisation model generator assisted with generative pre-trained transformer. *CoRR*, abs/2305.05811(arXiv:2305.05811), 2023. doi:10.48550/arXiv.2305.05811.

2 Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. doi:10.48550/arXiv.1409.0473.

3 Christian Bessiere, Clément Carbonnel, Anton Dries, Emmanuel Hebrard, George Katsirelos, Nadjib Lazaar, Nina Narodytska, Claude-Guy Quimper, Kostas Stergiou, Dimosthenis C. Tsouros, and Toby Walsh. Learning constraints through partial queries. *Artif. Intell.*, 319:103896, 2023. doi:10.1016/j.artint.2023.103896.

4 Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. URL: https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

5 Jaime G. Carbonell and Jade Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In W. Bruce Croft, Alistair Moffat, C. J. van Rijsbergen, Ross Wilkinson, and Justin Zobel, editors, *SIGIR '98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 24-28 1998, Melbourne, Australia*, SIGIR '98, pages 335–336, New York, NY, USA, 1998. ACM. doi:10.1145/290941.291025.

6 Dhivya Chandrasekaran and Vijay Mago. Evolution of semantic similarity - A survey. *ACM Comput. Surv.*, 54(2):41:1–41:37, February 2022. doi:10.1145/3440755.

7 Harrison Chase. Langchain, October 2022. URL: https://github.com/langchain-ai/langchain.

8 Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. doi:10.48550/arXiv.2107.03374.

9 Jens Claes, Bart Bogaerts, Rocsildes Canoy, Emilio Gamba, and Tias Guns. Zebratutor: Explaining how to solve logic grid puzzles. In Katrien Beuls, Bart Bogaerts, Gianluca Bontempi, Pierre Geurts, Nick Harley, Bertrand Lebichot, Tom Lenaerts, Gilles Louppe, and Paul Van Eecke, editors, *Proceedings of the 31st Benelux Conference on Artificial Intelligence (BNAIC 2019) and the 28th Belgian Dutch Conference on Machine Learning (Benelearn 2019), Brussels, Belgium, November 6-8, 2019*, volume 2491 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2019. URL: `https://ceur-ws.org/Vol-2491/demo96.pdf`.

10 Cathleen Cortis Mack, Caterina Cinel, Nigel Davies, Michael Harding, and Geoff Ward. Serial position, output order, and list length effects for words presented on smartphones over very long intervals. *Journal of Memory and Language*, 97:61–80, 2017. `doi:10.1016/j.jml.2017.07.009`.

11 Parag Pravin Dakle, Serdar Kadioglu, Karthik Uppuluri, Regina Politi, Preethi Raghavan, SaiKrishna Rallabandi, and Ravisutha Srinivasamurthy. Ner4opt: Named entity recognition for optimization modelling from natural language. In André A. Ciré, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 20th International Conference, CPAIOR 2023, Nice, France, May 29 - June 1, 2023, Proceedings*, volume 13884 of *Lecture Notes in Computer Science*, pages 299–319, Cham, 2023. Springer. `doi:10.1007/978-3-031-33271-5_20`.

12 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. `doi:10.18653/v1/n19-1423`.

13 Xuan-Dung Doan. VTCC-NLP at nl4opt competition subtask 1: An ensemble pre-trained language models for named entity recognition. *CoRR*, abs/2212.07219(arXiv:2212.07219), 2022. `doi:10.48550/arXiv.2212.07219`.

14 Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, Lei Li, and Zhifang Sui. A survey for in-context learning. *CoRR*, abs/2301.00234(arXiv:2301.00234), 2023. `doi:10.48550/arXiv.2301.00234`.

15 Eugene C. Freuder. Progress towards the holy grail. *Constraints An Int. J.*, 23(2):158–171, 2018. `doi:10.1007/s10601-017-9275-0`.

16 Eugene C. Freuder and Barry O'Sullivan. Grand challenges for constraint programming. *Constraints An Int. J.*, 19(2):150–162, 2014. `doi:10.1007/s10601-013-9155-1`.

17 Simon Frieder, Luca Pinchetti, Ryan-Rhys Griffiths, Tommaso Salvatori, Thomas Lukasiewicz, Philipp Christian Petersen, Alexis Chevalier, and Julius Berner. Mathematical capabilities of chatgpt. *CoRR*, abs/2301.13867, 2023. `doi:10.48550/arXiv.2301.13867`.

18 Neeraj Gangwar and Nickvash Kani. Highlighting named entities in input for auto-formulation of optimization problems. In Catherine Dubois and Manfred Kerber, editors, *Intelligent Computer Mathematics - 16th International Conference, CICM 2023, Cambridge, UK, September 5-8, 2023, Proceedings*, volume 14101 of *Lecture Notes in Computer Science*, pages 130–141. Springer, Springer, 2023. `doi:10.1007/978-3-031-42753-4_9`.

19 Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: program-aided language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 10764–10799. PMLR, PMLR, 2023. URL: `https://proceedings.mlr.press/v202/gao23f.html`.

20 Hila Gonen, Srini Iyer, Terra Blevins, Noah A. Smith, and Luke Zettlemoyer. Demystifying prompts in language models via perplexity estimation. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 10136–10148, Singapore, December 2023. Association for Computational Linguistics. `doi:10.18653/v1/2023.findings-emnlp.679`.

21 Tias Guns. Increasing modeling language convenience with a universal n-dimensional array, cppy as python-embedded example. In *Proceedings of the 18th workshop on Constraint Modelling and Reformulation at CP (Modref 2019)*, volume 19, 2019.

22 Jianglong He, Mamatha N, Shiv Vignesh, Deepak Kumar, and Akshay Uppal. Linear programming word problems formulation using ensemblecrf NER labeler and T5 text generator with data augmentations. *CoRR*, abs/2212.14657(arXiv:2212.14657), 2022. doi:10.48550/arXiv.2212.14657.

23 Adam Ishay, Zhun Yang, and Joohyung Lee. Leveraging large language models to generate answer set programs. In Pierre Marquis, Tran Cao Son, and Gabriele Kern-Isberner, editors, *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning, KR 2023, Rhodes, Greece, September 2-8, 2023*, KR '23, pages 374–383, 2023. doi:10.24963/kr.2023/37.

24 Elgun Jabrayilzade and Selma Tekir. Lgpsolver - solving logic grid puzzles automatically. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 1118–1123. Association for Computational Linguistics, 2020. doi:10.18653/v1/2020.findings-emnlp.100.

25 Sanghwan Jang. Tag embedding and well-defined intermediate representation improve auto-formulation of problem description. *CoRR*, abs/2212.03575(arXiv:2212.03575), 2022. doi:10.48550/arXiv.2212.03575.

26 Hyuhng Joon Kim, Hyunsoo Cho, Junyeob Kim, Taeuk Kim, Kang Min Yoo, and Sang-goo Lee. Self-generated in-context learning: Leveraging auto-regressive language models as a demonstration generator. *CoRR*, abs/2206.08082(arXiv:2206.08082), 2022. doi:10.48550/arXiv.2206.08082.

27 Zeynep Kiziltan, Marco Lippi, and Paolo Torroni. Constraint detection in natural language problem descriptions. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, IJCAI'16, pages 744–750. IJCAI/AAAI Press, 2016. URL: http://www.ijcai.org/Abstract/16/111.

28 Andrew K. Lampinen, Ishita Dasgupta, Stephanie C. Y. Chan, Kory W. Mathewson, Michael Henry Tessler, Antonia Creswell, James L. McClelland, Jane Wang, and Felix Hill. Can language models learn from explanations in context? In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Findings of the Association for Computational Linguistics: EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 537–563, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi:10.18653/v1/2022.findings-emnlp.38.

29 Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 7871–7880. Association for Computational Linguistics, Association for Computational Linguistics, 2020. doi:10.18653/v1/2020.acl-main.703.

30 Junyi Li, Tianyi Tang, Wayne Xin Zhao, and Ji-Rong Wen. Pretrained language model for text generation: A survey. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 4492–4499. ijcai.org, 2021. doi:10.24963/ijcai.2021/612.

31 Qingyang Li, Lele Zhang, and Vicky Mak-Hau. Synthesizing mixed-integer linear programming models from natural language descriptions. doi:10.48550/arXiv.2311.15271[math].

32 Xiaonan Li and Xipeng Qiu. Finding support examples for in-context learning. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 6219–6235, Singapore, December 2023. Association for Computational Linguistics. doi:10.18653/v1/2023.findings-emnlp.411.

33 Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for gpt-3? In Eneko Agirre, Marianna Apidianaki, and Ivan Vulic, editors, *Proceedings of Deep Learning Inside Out: The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures, DeeLIO@ACL 2022, Dublin, Ireland and Online, May 27, 2022*, pages 100–114, Dublin, Ireland and Online, May 2022. Association for Computational Linguistics. `doi:10.18653/v1/2022.deelio-1.10`.

34 Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Trans. Assoc. Comput. Linguistics*, 12:157–173, 2024. `doi:10.1162/tacl_a_00638`.

35 Michele Lombardi and Michela Milano. Boosting combinatorial problem modeling with machine learning. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 5472–5478. ijcai.org, July 2018. `doi:10.24963/ijcai.2018/772`.

36 Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 8086–8098, Dublin, Ireland, May 2022. Association for Computational Linguistics. `doi:10.18653/v1/2022.acl-long.556`.

37 Kostis Michailidis, Dimos Tsouros, and Tias Guns. CP-LLMs-ICL. Software, swhId: `swh:1:dir:5e4383ad6c4329796c9f21c51bbff4882dca8271` (visited on 2024-08-16). URL: `https://github.com/kostis-init/CP-LLMs-ICL`.

38 Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 11048–11064, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. `doi:10.18653/v1/2022.emnlp-main.759`.

39 Arindam Mitra and Chitta Baral. Learning to automatically solve logic grid puzzles. In Lluís Màrquez, Chris Callison-Burch, Jian Su, Daniele Pighin, and Yuval Marton, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1023–1033, Lisbon, Portugal, September 2015. The Association for Computational Linguistics. `doi:10.18653/v1/d15-1118`.

40 Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard CP modelling language. In Christian Bessiere, editor, *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, volume 4741 of *Lecture Notes in Computer Science*, pages 529–543. Springer, 2007. `doi:10.1007/978-3-540-74970-7_38`.

41 Yuting Ning, Jiayu Liu, Longhu Qin, Tong Xiao, Shangzi Xue, Zhenya Huang, Qi Liu, Enhong Chen, and Jinze Wu. A novel approach for auto-formulation of optimization problems. *CoRR*, abs/2302.04643(arXiv:2302.04643), 2023. `doi:10.48550/arXiv.2302.04643`.

42 OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023. `doi:10.48550/arXiv.2303.08774`.

43 Ganesh Prasath and Shirish Karande. Synthesis of mathematical programs from natural language specifications. *CoRR*, abs/2304.03287(arXiv:2304.03287), 2023. `doi:10.48550/arXiv.2304.03287`.

44 Steven Prestwich and Nic Wilson. A statistical approach to learning constraints. *International Journal of Approximate Reasoning*, page 109184, 2024. `doi:10.1016/j.ijar.2024.109184`.

45 Luc De Raedt, Andrea Passerini, and Stefano Teso. Learning constraints from examples. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial*

Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018, pages 7965–7970. AAAI Press, 2018. doi:10.1609/aaai.v32i1.12217.

46 Rindranirina Ramamonjison, Haley Li, Timothy T. L. Yu, Shiqi He, Vishnu Rengan, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. Augmenting operations research with auto-formulation of optimization models from problem descriptions. *CoRR*, abs/2209.15565(arXiv:2209.15565), 2022. doi:10.48550/arXiv.2209.15565.

47 Rindranirina Ramamonjison, Timothy T. L. Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. Nl4opt competition: Formulating optimization problems based on their natural language descriptions. In Marco Ciccone, Gustavo Stolovitzky, and Jacob Albrecht, editors, *NeurIPS 2022 Competition Track, November 28 - December 9, 2022, Online*, volume 220 of *Proceedings of Machine Learning Research*, pages 189–203. PMLR, PMLR, 2021. URL: https://proceedings.mlr.press/v220/ramamonjison22a.html.

48 Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H. Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 31210–31227. PMLR, 2023. URL: https://proceedings.mlr.press/v202/shi23a.html.

49 Helmut Simonis. Building industrial applications with constraint programming. In Hubert Comon, Claude Marché, and Ralf Treinen, editors, *Constraints in Computational Logics: Theory and Applications, International Summer School, CCL'99 Gif-sur-Yvette, France, September 5-8, 1999, Revised Lectures*, volume 2002 of *Lecture Notes in Computer Science*, pages 271–309. Springer, 1999. doi:10.1007/3-540-45406-3_6.

50 Dimosthenis C. Tsouros, Senne Berden, and Tias Guns. Guided bottom-up interactive constraint acquisition. In Roland H. C. Yap, editor, *29th International Conference on Principles and Practice of Constraint Programming, CP 2023, August 27-31, 2023, Toronto, Canada*, volume 280 of *LIPIcs*, pages 36:1–36:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.CP.2023.36.

51 Dimosthenis C. Tsouros, Senne Berden, and Tias Guns. Learning to learn in interactive constraint acquisition. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan, editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 8154–8162. AAAI Press, 2024. doi:10.1609/aaai.v38i8.28655.

52 Dimosthenis C. Tsouros, Hélène Verhaeghe, Serdar Kadioglu, and Tias Guns. Holy grail 2.0: From natural language to constraint models. *CoRR*, abs/2308.01589(arXiv:2308.01589), 2023. doi:10.48550/arXiv.2308.01589.

53 Giuseppe Vallar and Costanza Papagno. Phonological short-term store and the nature of the recency effect: Evidence from neuropsychology. *Brain and Cognition*, 5(4):428–442, 1986.

54 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, volume 30, pages 5998–6008, 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

55 Mark Wallace. Practical applications of constraint programming. *Constraints An Int. J.*, 1(1/2):139–168, 1996. doi:10.1007/BF00143881.

56 Kangxu Wang, Ze Chen, and Jiewen Zheng. Opd@nl4opt: An ensemble approach for the NER task of the optimization problem. *CoRR*, abs/2301.02459(arXiv:2301.02459), 2023. doi:10.48550/arXiv.2301.02459.

57 Xinyi Wang, Wanrong Zhu, Michael Saxon, Mark Steyvers, and William Yang Wang. Large language models are latent variable models: Explaining and finding good demonstrations for in-context learning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, volume 36, pages 15614–15638. Curran Associates, Inc., 2023. URL: http://papers.nips.cc/paper_files/paper/2023/hash/3255a7554605a88800f4e120b3a929e1-Abstract-Conference.html.

58 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. URL: http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

59 Jerry W. Wei, Jason Wei, Yi Tay, Dustin Tran, Albert Webson, Yifeng Lu, Xinyun Chen, Hanxiao Liu, Da Huang, Denny Zhou, and Tengyu Ma. Larger language models do in-context learning differently. *CoRR*, abs/2303.03846, 2023. doi:10.48550/arXiv.2303.03846.

60 Zhiyong Wu, Yaoxiang Wang, Jiacheng Ye, and Lingpeng Kong. Self-adaptive in-context learning: An information compression perspective for in-context example selection and ordering. In Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 1423–1436, Toronto, Canada, July 2023. Association for Computational Linguistics. doi:10.18653/v1/2023.acl-long.79.

61 Jiacheng Ye, Zhiyong Wu, Jiangtao Feng, Tao Yu, and Lingpeng Kong. Compositional exemplars for in-context learning. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 39818–39833. PMLR, 23–29 July 2023. URL: https://proceedings.mlr.press/v202/ye23c.html.

62 Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. Satlm: Satisfiability-aided language models using declarative prompting. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 45548–45580. Curran Associates, Inc., 2023. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/8e9c7d4a48bdac81a58f983a64aaf42b-Paper-Conference.pdf.

63 Xi Ye, Srinivasan Iyer, Asli Celikyilmaz, Veselin Stoyanov, Greg Durrett, and Ramakanth Pasunuru. Complementary explanations for effective in-context learning. In Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 4469–4484, Toronto, Canada, July 2023. Association for Computational Linguistics. doi:10.18653/v1/2023.findings-acl.273.

64 Ying-Jung Yvonne Yeh and Min-Hung Chen. Examining the primacy and recency effect on learning effectiveness with the application of interactive response systems (irs). *Technol. Knowl. Learn.*, 27(3):957–970, 2022. doi:10.1007/s10758-021-09521-6.

65 Yiming Zhang, Shi Feng, and Chenhao Tan. Active example selection for in-context learning. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 9134–9148, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi:10.18653/v1/2022.emnlp-main.622.

Table 4 Performance of various LLMs on LGPs using CP generation with R-MMR ($\lambda = 0.5$).

LLM	#Shots	#Err	Accuracy (%)		
			Model	Solution	Declaration
gpt-3.5-turbo-0125	4	6	71.00	74.00	87.98
gpt-4-0125-preview	4	16	61.00	64.00	73.85
deepseek-coder-33b-instruct	4	1	73.00	77.00	92.60
Mixtral-8x7B-Instruct-v0.1	4	19	52.00	52.00	70.01
Qwen1.5-72B-Chat	4	15	64.00	66.00	80.32
gpt-3.5-turbo-0125	8	4	74.00	77.00	90.49
gpt-4-0125-preview	8	10	77.00	82.00	85.87
deepseek-coder-33b-instruct	8	2	77.00	77.00	93.66
Mixtral-8x7B-Instruct-v0.1	8	18	58.00	61.00	74.90
Qwen1.5-72B-Chat	8	7	62.00	65.00	86.00

A Various LLMs

We also present preliminary results when running our methodology on LGPs for direct CP modelling using various LLMs in Table 4. We used the *OpenAI*[9], *Together AI*[10] and *DeepSeek*[11] APIs for these experiments. As a code-tuned LLM, the deepseek-coder-33b-instruct managed to produce the fewest models with errors. Additionally, gpt-4-0125-preview achieved the highest solution accuracy when prompted with 8 in-context examples.

B Model Equivalence Algorithm

To assert the equivalence of two constraint models, we first create a one-to-one mapping between their decision variables based on syntactic similarity. We then construct the conjunction of all constraints for each model. Assuming the two models are not equivalent, we add to a new combined model the negation of the equivalence formula between the two models, as defined in Equation (18). If the resulting model is unsatisfiable, the two models are considered equivalent. Otherwise, they are not. The detailed steps are provided in Algorithm 1.

C Errors in the generated CP model

C.1 0-shot

An example with a syntax error of a zero-shot attempt to directly model the first problem from the NL4Opt dataset with gpt-3.5-turbo-0125 is shown in Figure 5.

C.2 4-shot

An example with a syntax error on an LGP when prompting statically with 4 examples in the context is briefly shown in Figure 6.

[9] `https://platform.openai.com/docs/api-reference`
[10] `https://docs.together.ai/reference/chat-completions`
[11] `https://platform.deepseek.com/docs`

Algorithm 1 Model Equivalence.

```
1:  function ModelsEquivalent($m1$, $m2$)
2:      $vars1 \leftarrow$ GetVariables($m1$), $vars2 \leftarrow$ GetVariables($m2$)
3:      if length($vars1$) $\neq$ length($vars2$) then
4:          return False
5:      end if
6:      $mapping \leftarrow$ CreateMapping($vars1$, $vars2$)    ▷ Map variables based on syntax matching of their names
7:      $conj1 \leftarrow \bigwedge_{c \in m1.constraints} c$
8:      $conj2 \leftarrow \bigwedge_{c \in m2.constraints} c$
9:      $negatedImplications \leftarrow mapping \wedge ((conj1 \wedge \neg conj2) \vee (\neg conj1 \wedge conj2))$
10:     if IsUnsatisfiable($negatedImplications$) then
11:         return True    ▷ Models are equivalent
12:     else
13:         return False    ▷ Models are not equivalent
14:     end if
15: end function
```

```python
import cpmpy.solvers

# Define the variables
num_sled_dog_trips = intvar(0, 10)  # Number of sled dog trips
num_truck_trips = intvar(0, 10)  # Number of truck trips

# Define the objective function to maximize the number of fish transported
total_fish_transported = 100*num_sled_dog_trips + 300*num_truck_trips

# Define the constraints
cost_sled_dog_trips = 50*num_sled_dog_trips  # Cost of sled dog trips
cost_truck_trips = 100*num_truck_trips  # Cost of truck trips
total_cost = cost_sled_dog_trips + cost_truck_trips <= 1000  # Total cost
    constraint

# Number of sled dog trips must be less than the number of truck trips
num_sled_dog_trips <= num_truck_trips

# Define the LP problem
problem = cpmpy.Model(
    cpmpy.maximize(total_fish_transported),
    [total_cost]
)

# Solve the LP problem
cpmpy.solvers.cplex(problem)

# Error: module 'cpmpy' has no attribute 'maximize'
```

Figure 5 Generated code with syntax error. Configuration: gpt-3.5-turbo-0125, 0-shot.

```
from cpmpy import *

# Code omitted

# Clue 4: Of the runner who paid $9 and the contestant who ordered fettuccine,
      one was Margie and the other was Velma:
m += ((margie == price9) & (velma == fettuccine)) | ((margie == fettuccine) &
     (velma == price9)

# Error: ')' expected
```

Figure 6 Generated code with syntax error. Configuration: gpt-3.5-turbo-0125, 4-shot static ICL.

Table 5 Mixed CP Dataset. Configuration: gpt-3.5-turbo-0125, 4-shot static in-context examples selection.

Pathway	Solution Accuracy (%)
Direct	16.67
CP	50.00
+ BM	50.00
+ NER	44.44

D Mixed CP Dataset: Supplementary Results

Adding to the main results presented in the paper, Table 5 shows results for all four pathways and Table 6 displays the solution-level accuracy of the direct CP modelling pathway for all seven example retrieval strategies.

E Datasets Metadata

In this section, we present details of the datasets that we utilized in the experimental part. Table 7 describes the NL4Opt dataset. All values refer to the average number across each instance in the dataset. Regarding the LGPs, they contain on average 12 entities (or decision variables) and 4.55 clues per instance, while each instance has one unique solution. Table 8 outlines the type of each clue along with an example and its constraint representation. Finally, Table 9 provides some information about the instances of the mixed CP dataset that we curated.

Table 6 Mixed CP Dataset. Configuration: gpt-3.5-turbo-0125, 4 examples in the context, direct CP modelling pathway.

Strategy	Solution Accuracy (%)
Static	50.00
Random	38.89
SIM	55.56
R-SIM	55.56
MMR	61.11
R-MMR	55.56
LSRR	50.00

Table 7 NL4Opt information. To calculate the average number of optimal solutions for the train split, we did not take into consideration some outlier instances that had over 100 optimal solutions. All constraints and objectives are linear in this dataset. For more details please refer to the original paper [46].

Split (#)	#Decision Var.	#Constraints	#Obj. Terms	#Optimal Solutions
Train (713)	2.09	2.79	2.05	1.13
Test (289)	2.02	2.92	1.43	1.20

Table 8 LGPs clue types. In our CP models, we treat all puzzle entities as decision variables. Also refer to the other works employing LGPs [39, 24].

Type	Clue Example	`Constraint Expression (CPMpy [21])`
Equivalence	The Luzagueil is a chardonnay	`luzagueil == chardonnay`
XOR	The Annata Branco is either the 1992 wine or the syrah.	`Xor([annata == vintage1992, annata == syrah])`
Pairwise XOR	Of the pinot gris and the 1984 bottle, one is the Luzagueil and the other is the Zifennwein	`Xor([(pinot_gris == luzagueil) & (vintage1984 == zifennwein), (pinot_gris == zifennwein) & (vintage1984 == luzagueil)])`
AllDifferent	The four people are Deep Shadow, the superhero who started in 2007, the hero who started in 2009 and Matt Minkle	`AllDifferent([deep_shadow, _2007, _2009, matt_minkle])`
Arithmetic Comparison	The pinot gris was bottled 4 years after the merlot	`[((v1 == pinot_gris) & (v2 == merlot)) .implies(vintage_to_int[v1] == vintage_to_int[v2] + 4) for v1 in vintages for v2 in vintages]`

Table 9 Mixed CP Dataset information.

Instance	#Dec. Vars	#Const.	Optimisation	#Vars per Constraint	#(Optimal) Solutions
Five Floors	5	7	No	2	1
Bank Card	4	3	No	4	1
Guards and Apples	6	6	No	2	1
Magic Square	16	11	No	5	10
Thick as Thieves	6	7	No	3	1
Money Change	6	1	Yes	6	1
Colour Simple	6	9	Yes	2	>50
Movie Scheduling	9	26	Yes	2	3
Subset Sum	6	1	No	6	1
Subsets 100	20	4	No	15	>50
Maximal Ind. Sets	8	24	Yes	2	2
Exodus	20	9	No	10	17
People in a Room	17	13	No	9	>50
Kidney Exchange	64	24	Yes	8	1
Farmer and Cows	25	10	No	25	>50
Grocery	4	2	No	4	1
Climbing Stairs	20	22	No	12	>50
Hardy 1729	4	5	No	4	>50

F An LGP example

In Figure 7 we showcase an example from the LGPs dataset [39].

The Luzagueil is a chardonnay. The Annata Branco is either the 1992 wine or the syrah. The Friambliss is a syrah. Of the pinot gris and the 1984 bottle, one is the Luzagueil and the other is the Zifennwein. The pinot gris was bottled 4 years after the merlot. Vintages: 1984, 1988, 1992, 1996. Wines: Annata Branco, Friambliss, Luzagueil, Zifennwein. Types: chardonnay, merlot, pinot gris, syrah.

Decision Variables:
- Names of the wines: AnnataBranco, Friambliss, Luzagueil, Zifennwein (all different, 1 to 4)
- Vintages of the wines: Vintage1984, Vintage1988, Vintage1992, Vintage1996 (all different, 1 to 4)
- Types of the wines: Chardonnay, Merlot, PinotGris, Syrah (all different, 1 to 4)

Constraints:
- The Luzagueil is a chardonnay: Luzagueil == Chardonnay.
- The Annata Branco is either the 1992 wine or the syrah: Xor([AnnataBranco == Vintage1992, AnnataBranco == Syrah]).
- The Friambliss is a syrah: Friambliss == Syrah.
- Of the pinot gris and the 1984 bottle, one is the Luzagueil and the other is the Zifennwein: Xor([(PinotGris == Luzagueil, Vintage1984 == Zifennwein), (PinotGris == Zifennwein, Vintage1984 == Luzagueil)]).
- The pinot gris was bottled 4 years after the merlot: v1 == PinotGris and v2 == Merlot => v1_to_int == v2_to_int + 4, for all v1, v2 in vintages.

```python
from cpmpy import *

# Decision variables
# (variables with the same value will correspond to the same triplet of vintage,
    wine, and type)
# e.g. if annata == 1 and vintage1984 == 1 and chardonnay == 1, then Annata Branco
    is a 1984 chardonnay
annata, friambliss, luzagueil, zifennwein = wines = intvar(1, 4, shape=4)
vintage1984, vintage1988, vintage1992, vintage1996 = vintages = intvar(1, 4, shape
    =4)
chardonnay, merlot, pinot_gris, syrah = types = intvar(1, 4, shape=4)
# Integer representation, for comparison constraints
vintage_to_int = {vintage1984: 1984, vintage1988: 1988, vintage1992: 1992,
    vintage1996: 1996}  # in years

# Helper functions (for formulating comparison constraints)
def bottled_exactly_after_than(var1, var2, diff):
    """
    Formulate the constraint that var1 was bottled exactly diff years after var2.
    """
    return [((v1 == var1) & (v2 == var2)).implies(vintage_to_int[v1] ==
    vintage_to_int[v2] + diff) for v1 in vintages for v2 in vintages]

# Constraints
m = Model()
# All entities are different per category
m += AllDifferent(wines)
m += AllDifferent(vintages)
m += AllDifferent(types)
# Clue 1: The Luzagueil is a chardonnay:
m += luzagueil == chardonnay
# Clue 2: The Annata Branco is either the 1992 wine or the syrah:
m += Xor([annata == vintage1992,annata == syrah])
# Clue 3: The Friambliss is a syrah:
m += friambliss == syrah
# Clue 4: Of the pinot gris and the 1984 bottle, one is the Luzagueil and the other
     is the Zifennwein:
m += Xor([(pinot_gris == luzagueil) & (vintage1984 == zifennwein), (pinot_gris ==
    zifennwein) & (vintage1984 == luzagueil)])
# Clue 5: The pinot gris was bottled 4 years after the merlot:
m += bottled_exactly_after_than(pinot_gris, merlot, 4)
```

Figure 7 From top to bottom: Problem description, Blueprint Model, CPMpy model.