GALA: Global LLM Agents for Text-to-Model Translation

Junyang Cai[1], **Serdar Kadıoğlu**[2,3], **Bistra Dilkina**[1]
[1]Department of Computer Science, University of Southern California
[2]AI Center of Excellence, Fidelity Investments
[3]Department of Computer Science, Brown University
caijunya@usc.edu, serdark@cs.brown.edu, dilkina@usc.edu

Abstract

Natural language descriptions of optimization or satisfaction problems are challenging to translate into correct MINIZINC models, as this process demands both logical reasoning and constraint programming expertise. We introduce GALA, a framework that addresses this challenge with a global agentic approach: multiple specialized large language model (LLM) agents decompose the modeling task by global constraint type. Each agent is dedicated to detecting and generating code for a specific class of global constraint, while a final assembler agent integrates these constraint snippets into a complete MINIZINC model. By dividing the problem into smaller, well-defined sub-tasks, each LLM handles a simpler reasoning challenge, potentially reducing overall complexity. We conduct initial experiments with several LLMs and show better performance against baselines such as one-shot prompting and chain-of-thought prompting. Finally, we outline a comprehensive roadmap for future work, highlighting potential enhancements and directions for improvement.

1 Introduction

Building correct MINIZINC models from natural language descriptions is a complex challenge. Recently, Singirikonda et al. [2025] has introduced several modeling co-pilot frameworks as well as a supporting dataset, TEXT2ZINC, and corresponding leaderboard for benchmarking this task. Existing co-pilot evaluations on TEXT2ZINC (using direct prompting, chain-of-thought, and compositional strategies, and external tools such as NER4OPT [Kadıoğlu et al., 2024], Knowledge Graphs) found that even powerful LLMs are "not yet a push-button technology" for generating combinatorial models from text. In other words, general-purpose prompting often fails to capture all variables and constraints correctly, especially for harder optimization problems. This motivates research into more structured and guided methods.

One promising direction is to break the problem into manageable pieces. Multi-step or multi-agent frameworks have begun to emerge for NL4Opt [Ramamonjison et al., 2023] (natural language for optimization) tasks. For example, Chain-of-Experts by Xiao et al. [2023] assigns multiple LLM "experts" with specific roles (e.g. interpreting text, formulating model components, coding, verifying) coordinated by a central conductor. This cooperative agent approach significantly outperformed prior single-LLM methods on complex operations research problems. Similarly, the OptiMUS system by AhmadiTeshnizi et al. [2023] uses an LLM-based agent to iteratively identify parameters, write constraints, and debug a linear program model, achieving a higher problem-solving rate than basic one-shot prompting. These results suggest that decomposing the modeling task and giving LLMs more structured guidance can substantially improve performance.

Preprint.

The main drawback of existing multi-agent approaches is that each agent still inherits the full complexity of the problem rather than focusing on a narrower, more tractable sub-task. To address this, we propose a new agentic framework for translating text to MINIZINC [Nethercote et al., 2007] that is centered around global constraints. In Constraint Programming (CP), global constraints such as `all_different` and `cumulative` are high-level primitives that capture common patterns among variables. They offer expressive and efficient building blocks in models, and many optimization problems can be described as a combination of such global constraints.

Our approach, **GALA**: **G**lob**A**l **L**LM **A**gents leverages this by dedicating a specialized LLM agent to each type of global constraint, turning model generation into a collaboration of focused experts rather than one monolithic generation. More broadly, our approach can be viewed as aligning and combining the key strength of Constraint Programming (i.e., global constraints) with the Agentic Frameworks. The following sections outline background and related work, our framework, preliminary findings, and future roadmap.

2 Background and Related Work

2.1 HOLY GRAIL 2.0, NL4OPT NER4OPT, & OPTIMUS

LLMs are increasingly applied to optimization and constraint programming. Holy Grail 2.0 [Tsouros et al., 2023] outlined a blueprint for conversational modeling assistants. Early work Ramamonjison et al. [2023] tackled linear programming via entity recognition and logical-form translation, showing promising ChatGPT results on NL4OPT.NER4OPT showed that accuracy of LLM-generated MINIZINC improves with in-line entity annotations [Dakle et al., 2023] as also built into several co-pilot pipelines [Kadıoğlu et al., 2024]. Agentic methods followed: a multi-agent Chain-of-Experts [Xiao et al., 2023] and Optimus [AhmadiTeshnizi et al., 2023], a modular system for complex descriptions. Text-to-model translation was also explored via simple decomposition prompts with GPT [Tsouros et al., 2023]. Extending this, RAG-based in-context learning built CPMpy models [Michailidis et al., 2025]. Concurrently to our work, Szeider [2025] introduce a Reason-and-Act framework solving all 101 CP-Bench tasks [Michailidis et al., 2025].

2.2 MINIZINC & TEXT2ZINC

MINIZINC [Nethercote et al., 2007] is a high-level, solver-agnostic constraint modeling language for discrete and continuous satisfaction and optimization problems. It provides a rich library of global constraints that capture common CP patterns, allowing users to express problems declaratively rather than through low-level decompositions. MINIZINC's practical utility is enhanced by its clean separation between models and instances. Given these features and the availability of existing TEXT2ZINC[Singirikonda et al., 2025] datasets, we benchmark different approaches of NL4OPT on MINIZINC execution accuracy and solution accuracy.

2.3 CP & Global Constraints

In CP, global constraints concisely represent recurring patterns like all-differentness, resource limits, ordering, counting, etc., which are found across many scheduling, assignment, and configuration problems. For example, `all_different` enforces that a set of decision variables all have distinct values – a common requirement in scheduling and allocation problems. Other examples include `cumulative` (which enforces scheduling resource capacity over time) and `global_cardinality` (which limits how many variables take each value). Our approach GALA is designed to take advantage of CP global constraints in an agentic LLM architecture as detailed next.

3 GALA: Global LLM Agents

Specialized LLM Agents per Constraint: For each type of global constraint, we instantiate a separate LLM agent with a specialized prompt. This prompt primes the agent to act as an expert in detecting and formulating that constraint in MINIZINC. The agent receives the full problem description (and any input data) but its instructions are local: it should ignore the broader context and only answer the question, "Does this problem involve an <X> constraint? If yes, produce the MINIZINC snippet for it; if not, output FALSE." Each agent is effectively performing a binary classification (constraint present or not) followed by code generation for that constraint if needed.

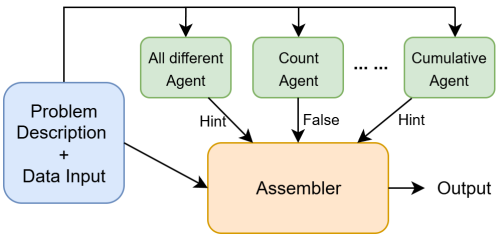

Figure 1: GALA: Global LLM Agents architecture for global-constraint detection and assembly.

Constraint Type	Detect (%)	False Detect (%)
`all_different`	88.1	14.42
`count`	67.9	28.30
`global_cardinality`	77.8	17.43
`cumulative`	58.3	17.59
`circuit`	100.0	2.75
`increasing`	78.6	4.67
`lex_less`	71.4	6.60

Table 1: Performance of GALA on detection and false-detection rates by global constraint type.

Critically, the agent is instructed not to produce any other modeling elements beyond its constraint. For instance, the `all_different` agent is prompted: "You are a MINIZINC modeling assistant specialized in detecting and modeling `all_different` constraints. Given a problem description, decide whether it requires one or more `all_different` constraints. If it does, generate only MINIZINC code specifying the `all_different` constraint and its variables. If it does not, return FALSE with a brief reason.". Similar templates are crafted for each global constraint type, incorporating definitions and common clue words (e.g. phrases like "each ... different" hint at `all_different`, or "no overlap" hints at `cumulative`). By isolating each agent's focus, **our main novelty is to simplify the reasoning task leveraging CP global constraints**. *Unlike the previous agentic approaches, our agents do not need to understand the entire problem structure, only whether a specific pattern appears and how to encode it.*

Assembler LLM for Model Integration: Once the constraint-specific agents have each returned either a code snippet or FALSE, an assembler LLM takes over. The assembler's input includes the original problem description and all the constraint snippets (we call them "hints") provided by the agents that found a constraint. The assembler's role is to compile a complete, coherent MINIZINC model from these pieces. We prompt the assembler agent as if it were "the world's best MINIZINC programmer" tasked with integrating hints and filling in the gaps. Concretely, it must: (1) declare all decision variables (and their domains) that are needed, possibly renaming or merging variables from different snippets for consistency; (2) analysis and decide if to include provided global constraints or not; (3) add any remaining constraints from the text that were not covered by the hints; (4) determine the objective (if an optimization problem) or a satisfy goal, based on the description; and (5) append a proper solve item and output format. The assembler may ignore an irrelevant hint if an agent was mistaken, but in general, it tries to use all valid snippets. By design, this agent has the most complex task, since it sees the full problem and must ensure completeness. However, because much of the heavy lifting (expressing complex constraints) is done by the specialized snippets, the assembler can focus on glueing components together and writing boilerplate code.

Figure 1 presents the overall architecture which is essentially a team of specialists: each constraint agent *independently* proposes part of the model, and the assembler (acting as the principal architect) reviews and integrates these contributions into the final solution. This approach aims to reduce the cognitive load on any single LLM. Eech agent works on a local model of the structured sub-problem, thanks to CP global constraints, instead of the entire text-to-model translation.

4 Initial Results

We conduct an initial evaluation of GALA, focusing on two aspects: (1) the ability of global agents to correctly detect global constraints, and (2) the end-to-end performance of the agentic pipeline compared to baseline prompting strategies and chain-of-thought (CoT) [Wei et al., 2022].

4.1 Performance of Global Agents

Table 1 reports detection performance for seven global constraint types using the open-source LLM Phi4. We evaluate on all 567 instances from the TEXT2ZINC dataset [Singirikonda et al., 2025]. For each constraint type, the detection rate is computed over the subset of instances whose ground-truth model contains that constraint. As shown in the table, the average detection rates is around 70% to 80%. To estimate the false detection rate, we use 110 verified TEXT2ZINC instances and exclude those whose ground-truth model includes the target constraint.

Model	Strategy	Execution Rate (count)	Solve Rate (count)	Avg Score
o3-mini	GALA	**57.27% (63)**	**32.73% (36)**	**45.00%**
o3-mini	CoT	52.73% (58)	30.91% (34)	41.82%
gpt-4o-mini	GALA	**33.64% (37)**	**17.27% (19)**	**25.45%**
gpt-4o-mini	CoT	31.82% (35)	12.73% (14)	22.27%
gpt-oss:20b	GALA	**17.27% (19)**	8.18% (9)	12.73%
gpt-oss:20b	CoT	16.36% (18)	**10.00% (11)**	**13.18%**
gpt-oss:20b	baseline	11.81% (13)	7.27% (8)	9.54%

Table 2: Execution rate, solve rate (numbers of executed/solved instances in parentheses), and average score across 110 TEXT2ZINC instances.

As shown Table 1, overall, our agents achieve strong detection rates, and false detection rates are generally low for rarer constraints; the main exception is `count` (28.3%), suggesting that distinguishing counting patterns from other numerical constraints remains a challenge. These results indicate that the individual agents are reasonably effective, with room for improvement through stronger base models or prompt optimization.

4.2 Performance of GALA on Text-to-Model Translation

In Table 2, we compare GALA with direct prompting (baseline) and chain-of-thought (CoT) across three LLM configurations: OpenAI o3-mini [OpenAI, 2025b], gpt-4o-mini [OpenAI, 2023], and the open-source gpt-oss 20B model [OpenAI, 2025a]. We execute each model and strategy on 110 verified instances from the TEXT2ZINC datasets [Singirikonda et al., 2025].

Agents consistently outperform CoT on the stronger models (o3-mini, GPT-4o-mini) across execution, solve, and mean score, and remain competitive on the 20B open model. While CoT gives a clear lift over the non-agent baseline on 20B, our first-pass *global agents*, with no model tuning, prompt optimization, or hyperparameter search, match or slightly exceed CoT on the stronger models, and substantially improve execution success overall. This supports the claim that the gains come from the decomposition/agentic assembly itself rather than prompt or parameter scaling.

5 Future Work

Optimize Global Agents: Replace hand-crafted prompts with systematic optimization (automatic search, curated few-shot exemplars, adversarial negatives) and consider fine-tuning per global constraint to boost precision and recall. Pair these with compile-time snippet validation so each agent not only detects constraints correctly but also emits syntactically valid MINIZINC.

Unblock the Assembler: Add a supervisor to extract variables and objectives before delegation, or a post-hoc linker to unify names and deduplicate constraints. Build a systematic error taxonomy to map where agents and the assembler succeed or fail, driving targeted fixes and feedback loops that patch prompts or trigger regenerations, improving local correctness and global assembly coherence.

Scale Evaluation: Go beyond small models (e.g., Phi-4) by running stronger LLMs (e.g., GPT-4) and sweeping both open- and closed-weight models across sizes. Benchmark on datasets rich in global-constraint instances, currently 70% of TEXT2ZINC lacks them, to better showcase the approach and reveal architectural headroom.

6 Conclusion

We introduce **GALA**: Global LLM Agents for Text-to-Model translation of satisfaction and optimization problems. We leverage specialized LLM agents to global constraints and an assembler to integrate them. Even in a first pass, without tuning, prompt engineering, or hyperparameter search, GALA outperforms a carefully curated CoT on stronger models and improve executability on the open 20B model, indicating that decomposition and agentic assembly are the key drivers of gains. The approach is modular and immediately extensible: finer handling of subtle constraints, compile-aware validation of snippets, and more robust staged assembly (conflict resolution, retry, and self-rectification) are natural next steps. With systematic error analysis and targeted feedback loops, we expect to turn this prototype into a robust and strong default for modeling co-pilots.

Acknowledgment

The National Science Foundation (NSF) partially supported the research under grant #2112533: "NSF Artificial Intelligence Research Institute for Advances in Optimization (AI4OPT)" and grant #2346058: "NRT-AI: Integrating Artificial Intelligence and Operations Research Technologies".

References

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. Optimus: Optimization modeling using mip solvers and large language models. *arXiv preprint arXiv:2310.06116*, 2023.

Parag Pravin Dakle, Serdar Kadıoğlu, Karthik Uppuluri, Regina Politi, Preethi Raghavan, SaiKrishna Rallabandi, and Ravisutha Srinivasamurthy. Ner4opt: Named entity recognition for optimization modelling from natural language. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 299–319. Springer, 2023.

Serdar Kadıoğlu, Parag Pravin Dakle, Karthik Uppuluri, Regina Politi, Preethi Raghavan, SaiKrishna Rallabandi, and Ravisutha Srinivasamurthy. Ner4opt: named entity recognition for optimization modelling from natural language. *Constraints*, pages 1–39, 2024.

Kostis Michailidis, Dimos Tsouros, and Tias Guns. Cp-bench: Evaluating large language models for constraint modelling. *arXiv preprint arXiv:2506.06052*, 2025.

Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In *International conference on principles and practice of constraint programming*, pages 529–543. Springer, 2007.

OpenAI. Gpt-4 technical report. 2023. URL `https://arxiv.org/abs/2303.08774`.

OpenAI. gpt-oss-120b & gpt-oss-20b model card, August 2025a. URL `https://cdn.openai.com/pdf/419b6906-9da6-406c-a19d-1bb078ac7637/oai_gpt-oss_model_card.pdf`. Model card.

OpenAI. Openai o3 and o4-mini system card, April 2025b. URL `https://cdn.openai.com/pdf/2221c875-02dc-4789-800b-e7758f3722c1/o3-and-o4-mini-system-card.pdf`. System card.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, et al. Nl4opt competition: Formulating optimization problems based on their natural language descriptions. In *NeurIPS 2022 competition track*, pages 189–203. PMLR, 2023.

Akash Singirikonda, Serdar Kadioglu, and Karthik Uppuluri. Text2zinc: A cross-domain dataset for modeling optimization and satisfaction problems in minizinc. *arXiv preprint arXiv:2503.10642*, 2025.

Stefan Szeider. Cp-agent: Agentic constraint programming. *arXiv preprint arXiv:2508.07468*, 2025.

Dimos Tsouros, Hélène Verhaeghe, Serdar Kadıoğlu, and Tias Guns. Holy grail 2.0: From natural language to constraint models. *arXiv preprint arXiv:2308.01589*, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-Thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022. URL `https://arxiv.org/abs/2201.11903`.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, et al. Chain-of-experts: When llms meet complex operations research problems. In *The twelfth international conference on learning representations*, 2023.