Text2Model: Copilots, Datasets and Fine-Tuning Small Language Models

Master Class on LLMs for Combinatorial Reasoning, CPAIOR, 2026

Serdar Kadıoğlu

Adj. Assoc. Professor, Dept. of Computer Science, Brown

Group VP, AI Center of Excellence, Fidelity Investments

BROWN

skadio.github.io

Learning & Reasoning

Data Science: ML/DL/NLP/LLMs/etc.

Focuses on **machine learning using historical data** to identify patterns and make predictions. Excels at pattern recognition, classification, and forecasting.

System I - Predictive Models

- Learning from historical data patterns
- Probabilistic predictions and insights
- Ideal for unstructured problems
- Applications include recommendation systems, image recognition, and natural language processing

Decision Science: OR/MIP/CP/SAT/LS/etc.

Focuses on **combinatorial satisfaction and optimization** using logical and mathematical models. Provides provable optimality and explicit reasoning.

System II - Prescriptive Models

- Mathematical and logical formulations
- Provably optimal for deterministic environments
- Perfect for structured problems
- Applications include verification, planning, scheduling, routing, and resource allocation

Learning & Reasoning

Data Scie **AT/LS/etc.**

Focuses on **nd optimization**
to identify p rovides provable
pattern reco

System I –

- Learning
- Probabili ironments
- Ideal for
- Applicati ning,
  image re cation

Evolution of AI Paradigms

Classical AI → **Modern AI** → **Generative AI**

expert systems (1950 – 2010)

supervised learning (2010 – 2020)

self-supervision (2020+)

If-Then Rules

Deep Learning

Large Language Models

Significant increase in investment, research, accessibility & visibility over the recent years

The Evolution of AI Paradigms: From Classical AI to Modern and Generative AI (AAAI YouTube)

Strategic Pillars of Enterprise AI @ Fidelity AI Center

AI Learning from Offline Data

Robust, scalable, reproducible features from structured, unstructured, and semi-structured datasets.

Selective, TextWiser, Seq2Pat

AI for Learning from Online Feedback

Adaptive, real-time, A/B testing systems that continuously learn from user interaction.

Mab2Rec, MABWiser

AI for Decision Making

Large-scale, integrated, (meta) solvers for resource management and optimization.

Forge, Balans, PathFinder

AI for Automated Assistants

Extraction and translation of natural language into downstream tasks and intents for human-computer interaction.

Text2Model, Text2Zinc, Learn2Zinc, Gala, Ner4Opt, iCBS

Responsible AI

Horizontal capabilities for explainability, evaluation, fairness, and bias mitigation across all systems.

Jurity, BoolXAI, BoolXLLM

Strategic Pillars of Enterprise AI @ Fidelity AI Center

AI Learning fro

Robust, scalable, reproc
from structured, unstru
semi-structured datase

Selective, TextWise

AI for Decision Making

e-scale, integrated, (meta)
ers for resource
agement and optimization.

orge, Balans, PathFinder

AI for Automate

Extraction and translatio
tasks and intents for hu

Gala, Ner

inability, evaluation, fairness,
stems.

oolXAI

Received: 26 July 2025 | Revised: 27 August 2025 | Accepted: 2 September 2025

DOI: 10.1002/aaai.70032

SPECIAL TOPIC ARTICLE

Open-source AI at scale: Establishing an enterprise AI strategy through modular frameworks

Serdar Kadıoğlu[1,2]

[1]AI Center of Excellence, Fidelity Investments, Boston, Massachusetts, USA

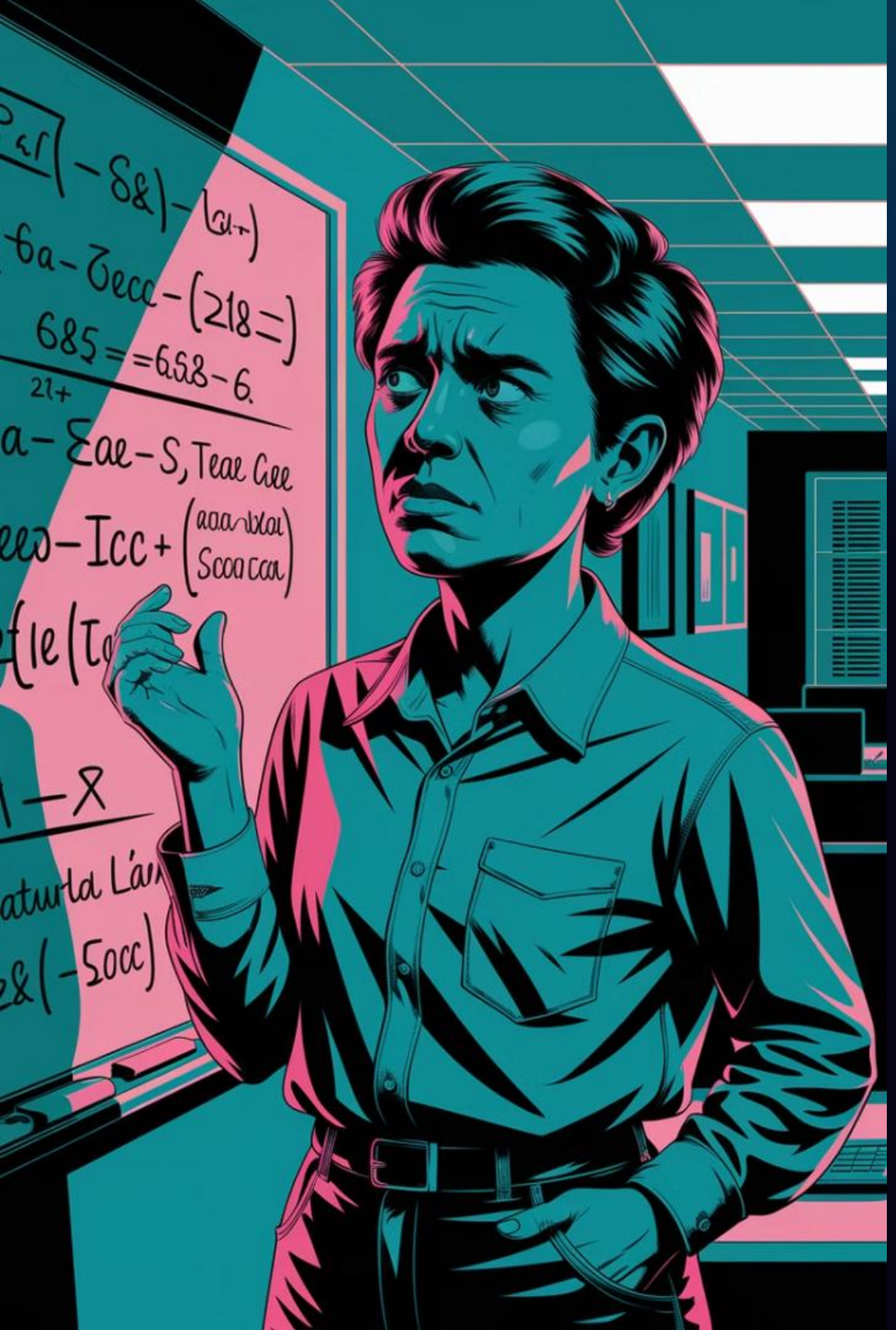
[2]Department of Computer Science, Brown University, Providence, Rhode Island, USA

Correspondence
Serdar Kadıoğlu
Email: serdark@cs.brown.edu

Abstract

We present a comprehensive enterprise AI strategy developed within the AI Center of Excellence at Fidelity Investments, emphasizing the strategic integration of open-source AI frameworks into scalable, modular, and reproducible enterprise-grade solutions. Our approach is structured around five key pillars: learning from offline data, learning from online feedback, intelligent decision-making, automated assistants, and responsible AI practices. Through a suite of 12 open-source libraries, we demonstrate how modular and interoperable tools can collectively enhance scalability, fairness, and explainability in real-world AI deployments. We further illustrate the impact of this strategy through three enterprise case studies. Finally, we distill a set of best deployment practices to guide organizations in implementing modular, open-source AI strategies at scale.

Open-Source AI at Scale: Establishing an Enterprise AI Strategy [AI Magazine'25]



The De-Facto Model-and-Run Strategy

1

Problem Description

Users **describe optimization problems** in natural language, which contains ambiguous references to variables, constraints, and objectives that must be precisely identified.

2

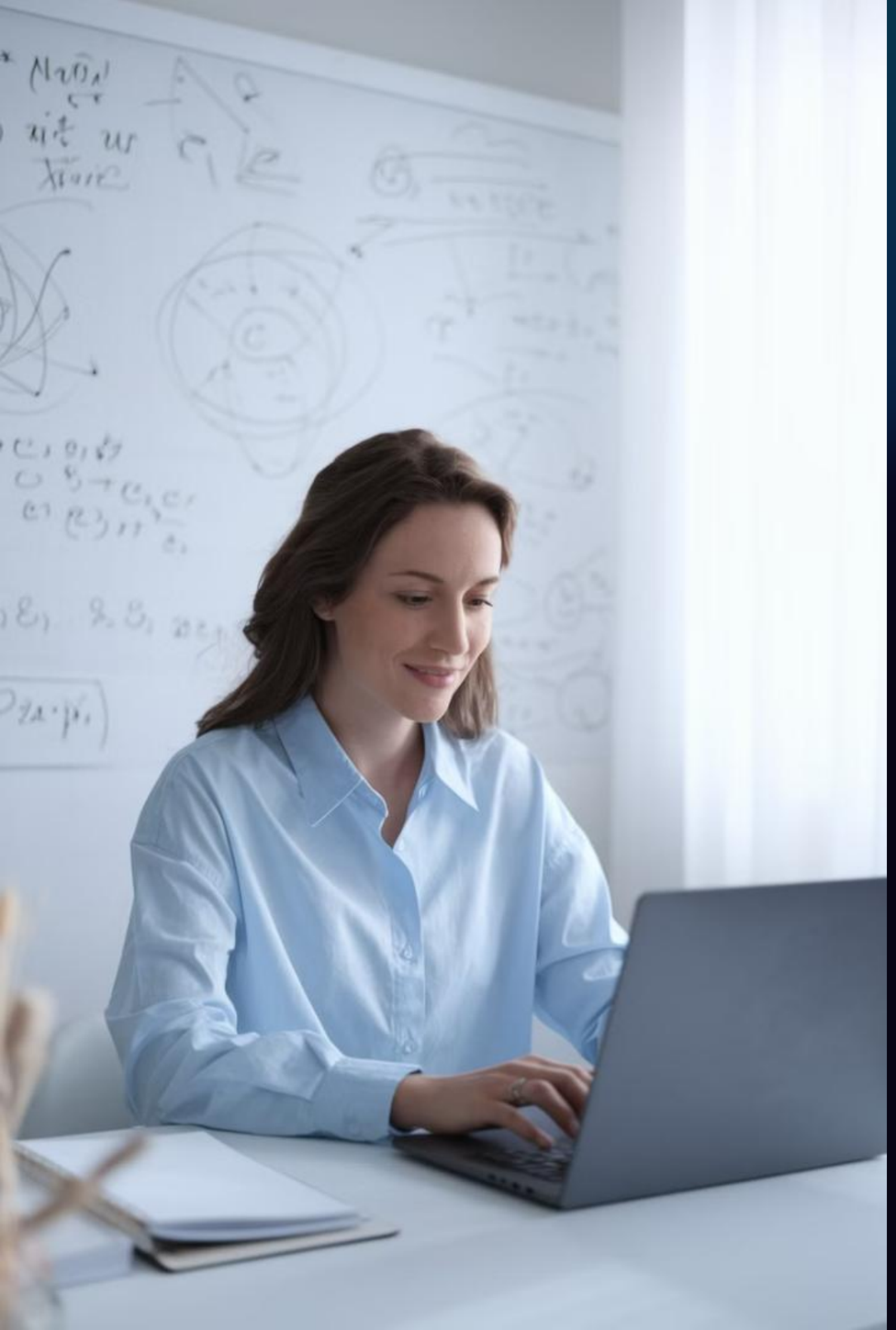
Model Formulation

Experts must **manually transform problem descriptions** into formal mathematical models, a process that requires specialized knowledge and is prone to errors.

3

Solution Finding

Once properly modeled, optimization solvers can find optimal solutions, but the **modeling barrier** remains a significant obstacle to wider adoption of optimization technology.



Decision Making in the Era of Large-Language Models

1 Reasoning: Optimization

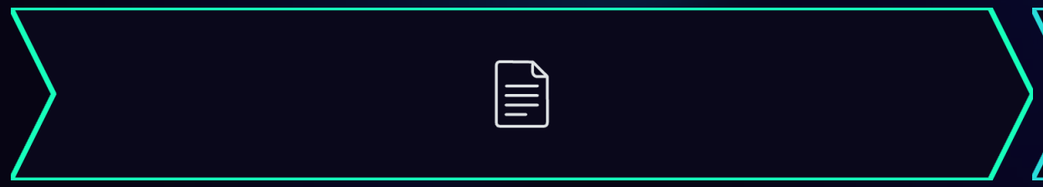
- Optimization technology and constraint solving techniques are powerful and have many applications.
- The cognitive barrier of translating problem descriptions into formal constraint models persists.

2 Learning: Large-Language Models

- LLMs have found success in many fields recently.
- However, they still face challenges in generating constraint models from free-form natural language text.

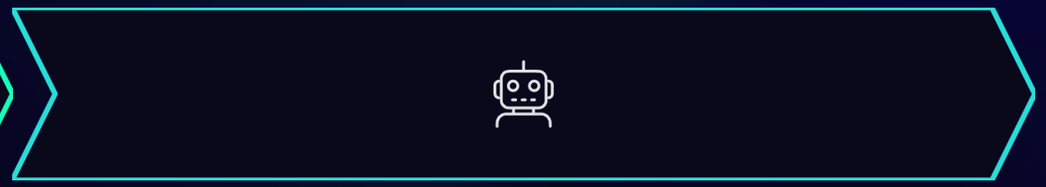
Our Vision: Modeling Co-Pilots

A paradigm shift integrating **automated modeling assistants** capable of translating natural language into formal optimization formulations.



Natural Language

Problem descriptions in free-form text



LLM Co-Pilot

Automated translation and formulation

Formal Models

Executable constraint model code

Solution | Interactivity | Feedback Loop

Verified results

LLM Copilots for Text-to-Model Translation

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

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

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

Paper | Copilots | Slides | Leaderboard | Interactive Editor | Collections

https://skadio.github.io/text2model

Our Contributions

Text2Zinc Dataset

A **unified cross-domain dataset** curated to work with **LLM co-pilots** and **interactive editor** *Singirikonda et. al., AAAI'25*

Text2Model Copilots

A suite of **LLM Modeling copilots** with multiple strategies of varying complexity and **leaderboard** *Kadioglu et. al*

Gala: Global LLM Agents

A **global agentic approach** with specialized agents decompose modeling by global constraint. *Cai et. al. NeurIPS'25*

Learn2Zinc

Targeted **fine-tuning** to teach small language models to generate syntactically correct MiniZinc *Kadioglu et. al*

Ner4Opt

Extracting optimization components from free-form natural language text. *Kadioglu et. al, Constraints'24*

Holy Grail 2.0

Blueprint for **optimization modelling co-pilots** with feedback loop and user interactivity *Tsouros et. al., 2023*

Definition: Combinatorial Problems

CSPs: Constraint Satisfaction Problem

A Constraint Satisfaction Problem is defined as a triple:

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

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

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

COPs: Constrained Optimization Problems

Optimization augment CSPs with an objective function:

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

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

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

Definition: Modeling Co-Pilots

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

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

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

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

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

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

[2604.12955] Modeling Copilots for Text-to-Model Translation, S. Kadioglu et. al.

Text-to-Model Translation

Text2Zinc Dataset

A **unified cross-domain dataset** curated to work with **LLM co-pilots** and **interactive editor** *Singirikonda et. al., AAAI'25*

Text2Model Copilots

A suite of **LLM Modeling copilots** with multiple strategies of varying complexity and **leaderboard** *Kadioglu et. al*

Gala: Global LLM Agents

A **global agentic approach** with specialized agents decompose model global constraint *Cai et. al. NeurIPS'25*

Learn2Zinc

Targeted **fine-tuning** to teach small language models to generate syntactically correct MiniZinc *Kadioglu et. Al*

Ner4Opt

Extracting optimization components from free-form natural language text. *Kadioglu et. al, Constraints'24*

Holy Grail 2.0

Blueprint for **optimization modelling co-pilots** with feedback loop and user interactivity *Tsouros et. al., 2023*

Text2Zinc: Motivation

Driving Progress

Datasets and benchmarks **fuel progress** in various domains: Computer Vision, NLP, and SAT, CP, MIP, RecSys, etc.

Room for Improvement

Current problem datasets have **potential for improvement** for integration with language models.

Structured Information & Metadata

Models and natural language descriptions of problems have been documented heavily but seldom occur together. Crucial **metadata is unavailable**.

Some Existing Resources

NL4OPT

- Linear programming problems
- No separation between problem description and data
- Relatively easy instances

NLP4LP

- Extends NL4OPT
- Introduces MIPs
- Evaluated with GurobiPy and cvxpy

ComplexOR & MAMO

- Standard OR Problems
- Evaluated with GurobiPy/COPT
- IndustryOR, Mamo

Optimization

Logic Grid Puzzles

- Introduces satisfaction problems in the form of logic grid puzzles

CSPLib

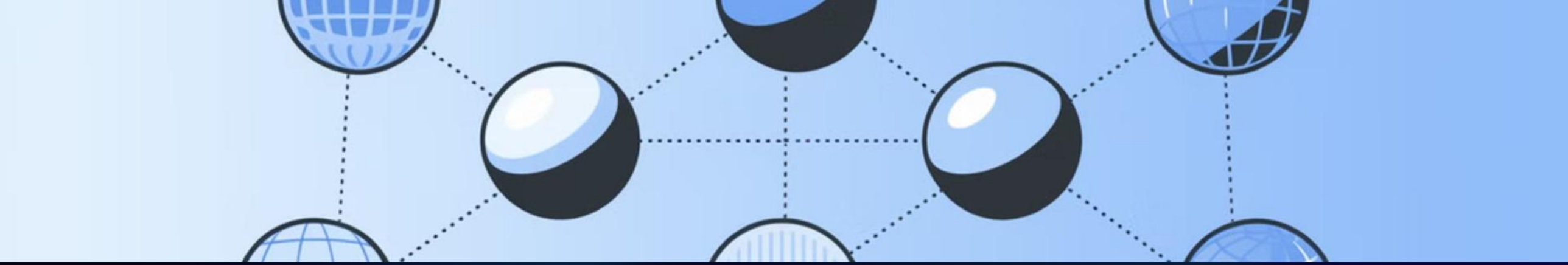
- CP and Satisfaction problems
- Not designed to work with ML or LLMs

Hakank's Models

- Extensive set of constraint models in various languages
- Does not capture metadata

Satisfaction

** Massive thank you to the community for contributing these valuable resources!*



Text2Zinc: Addressing Dataset Gaps

1 2 3 4

Cross-Domain

- Focus on combining both **optimization & satisfaction** problems.
- **Unified** dataset to incorporate LP, MIP, and CP problems with integer, Boolean, and continuous

Unified Format

- Unifies existing datasets.
- **Clear separation** of problem description & instance data.

Solver Agnostic

- Enables **solver agnostic** approaches.
- MIP, CP, SAT, LCG through MiniZinc.

Data Augmentation

- Clear and concise descriptions.
- Input and output specification.
- **Metadata** generation.
- Manual verification.

S. Kadıoğlu et. al. Text2Zinc: A Cross-Domain Dataset for Modeling Optimization and Satisfaction Problems in MiniZinc

MiniZinc: Solver-Agnostic Modeling

High-Level Language

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

Multiple Backends

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

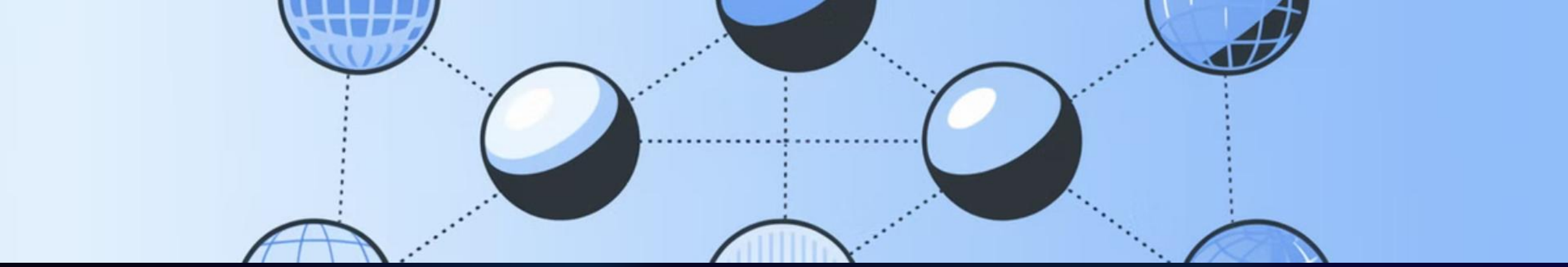
Global Constraints

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

MiniZinc Example: All Different Constraint

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

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

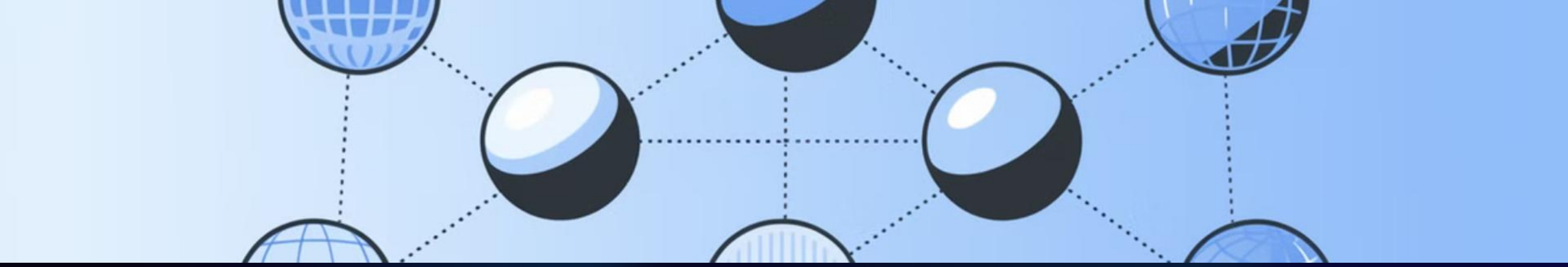
Global constraints allow users to **leverage higher-level abstractions** rather than focusing on low-level decomposition, significantly simplifying the modeling process.



Text2Zinc: Example Timetabling Problem – Description


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

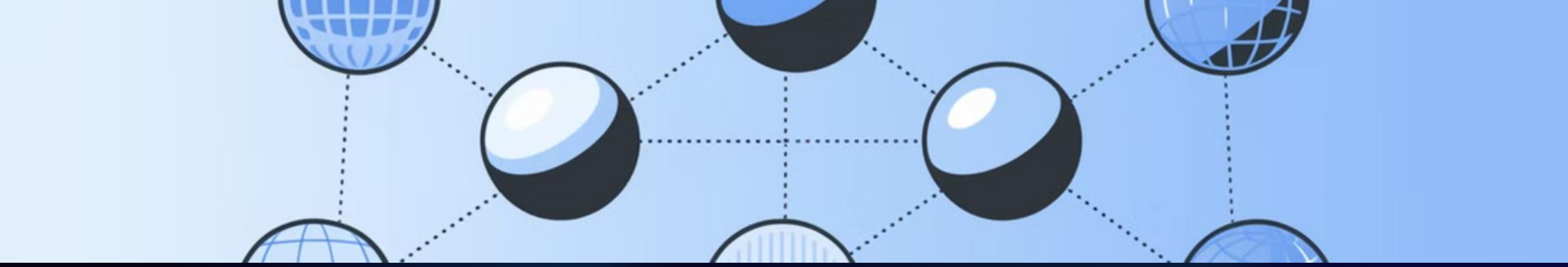

Figure 2 An example input with description, parameters, metadata, and output fields.



Text2Zinc: Example Timetabling Problem – Model

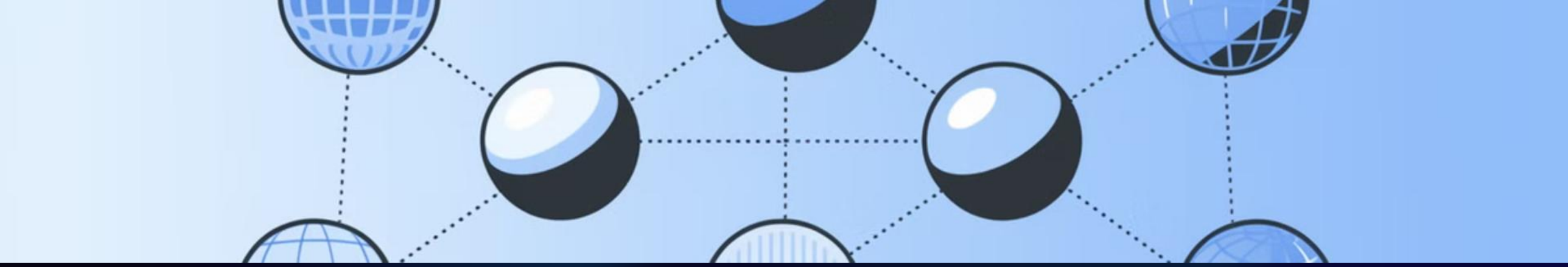
model.mzn

```
include "globals.mzn";

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

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

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



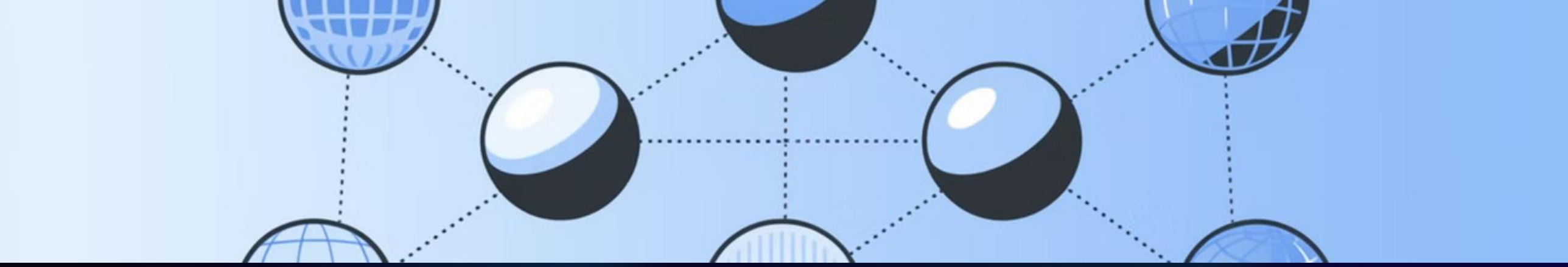
Text2Zinc: Example Timetabling Problem – Model

```
constraint
   % 1. Conflicts: Courses with common students must not be scheduled at the same time
   forall(c1, c2 in 1..courses where c1 < c2) (
      if conflict[c1, c2] = 1 then
         forall(p in 1..periods) (
            timetable[c1, p] + timetable[c2, p] <= 1
         )
      else
         true
      endif
   )
   % 2. Availabilities: Courses can only be scheduled in available periods
   /\
   forall(c in 1..courses, p in 1..periods) (
      if available[c, p] = 0 then
         timetable[c, p] = 0
```



Text2Zinc: Example Timetabling Problem – Model

```
% 3. Rooms: At most `rooms` lectures can be scheduled per period
/\
forall(p in 1..periods) (
   sum([timetable[c, p] | c in 1..courses]) <= rooms
)
% 4. Number of lectures per course must match the requirement
/\
forall(c in 1..courses) (
   sum([timetable[c, p] | p in 1..periods]) = requirement[c]
);
```



Text2Zinc: Example Timetabling Problem – Input & Output

data.dzn

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

output.json


```
{
"timetable": [
   [0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0,
      1, 0, 1, 0, 0, 0, 0, 0, 0],
   [1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
      0, 1, 0, 1, 1, 1, 1, 1, 1],
   [0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1,
      1, 0, 1, 1, 1, 1, 0, 1, 1],
   [0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1,
      0, 0, 0, 0, 0, 0, 0, 0, 0],
   [1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 1, 0, 0, 0, 0, 1, 0, 0]
 ]
 }
```

Text2Zinc Statistics

1775

Total Problems

Natural language instances

110

Manually Verified

High-quality curated data

11

Problem Domains

Diverse application areas covered

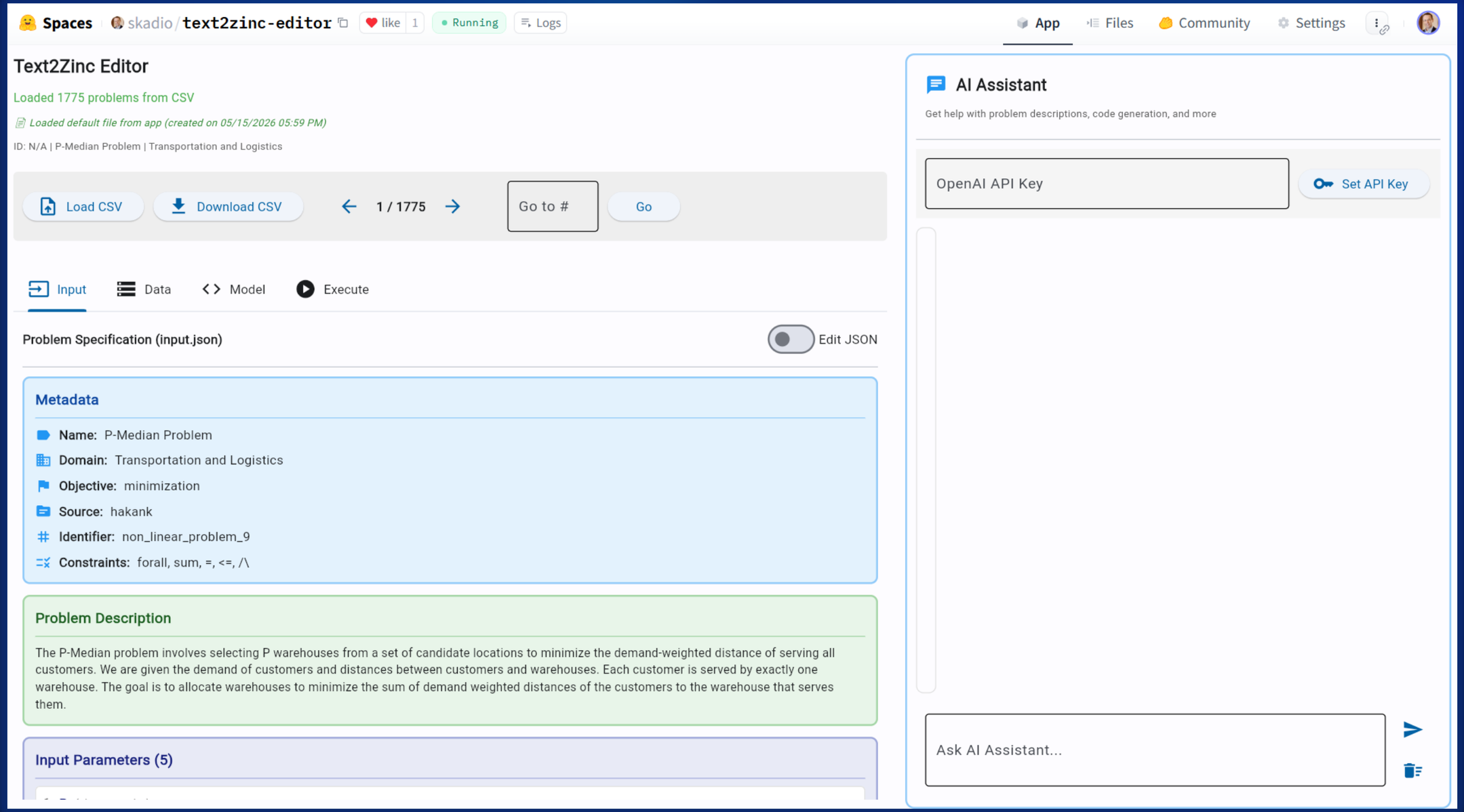
Our dataset includes instances of mixed of **LP, MIPs, and CP problems** across **7 different sources**
Providing a comprehensive benchmark for natural language to constraint model translation.

Problem Domain Coverage (Verified)

Domain	# Problems
Manufacturing & Production	22
Transportation & Logistics	15
Mathematical Modeling	14
Scheduling	13
Economic & Business Decisions	11
Puzzles & Games	7
Energy & Natural Resources	6
Healthcare & Human Systems	4
Finance & Investment	4
Network & Security	3
Industrial Engineering & Design	3

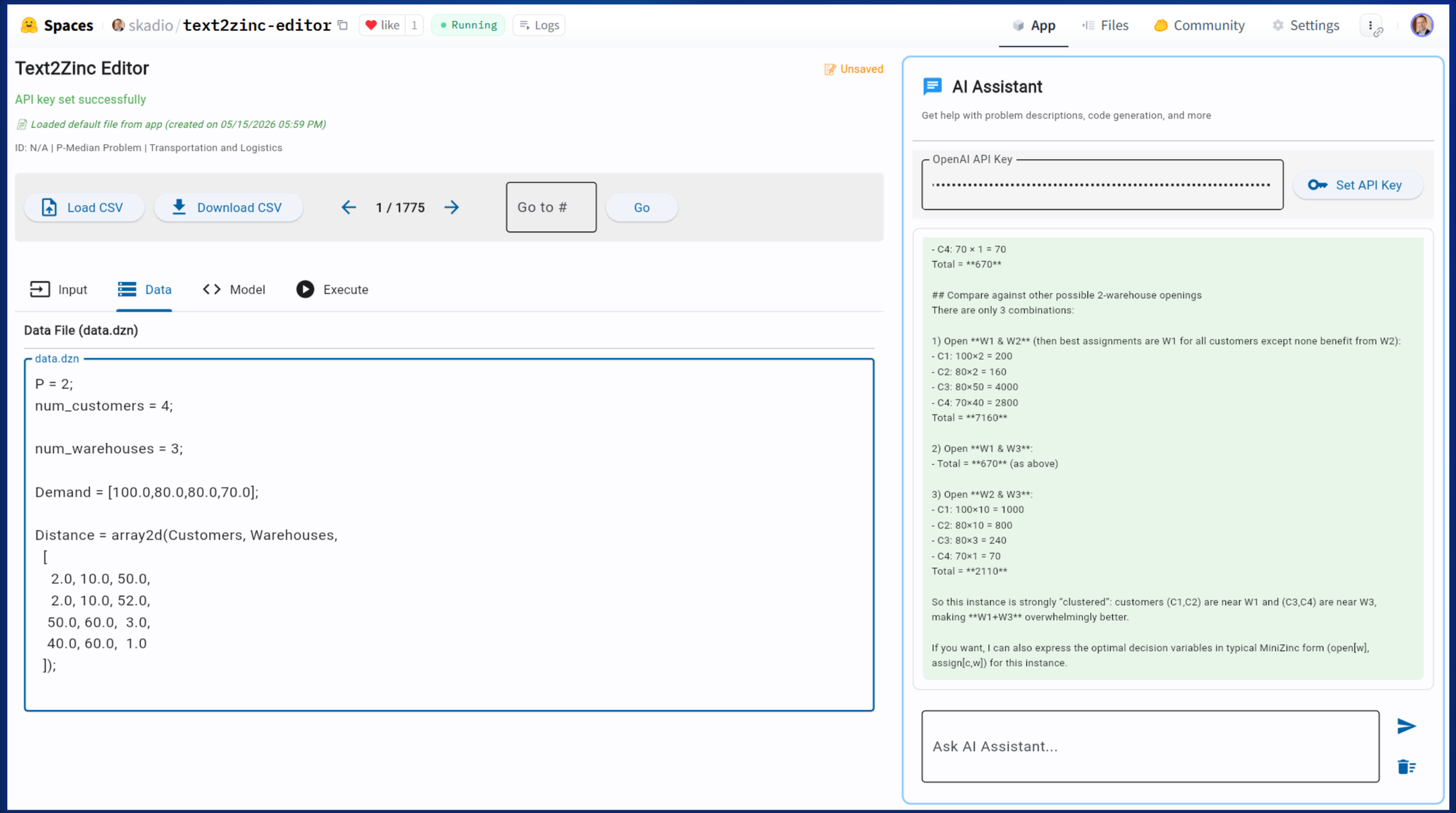
Text2Zinc Interactive Editor

https://huggingface.co/spaces/skadio/text2zinc-editor



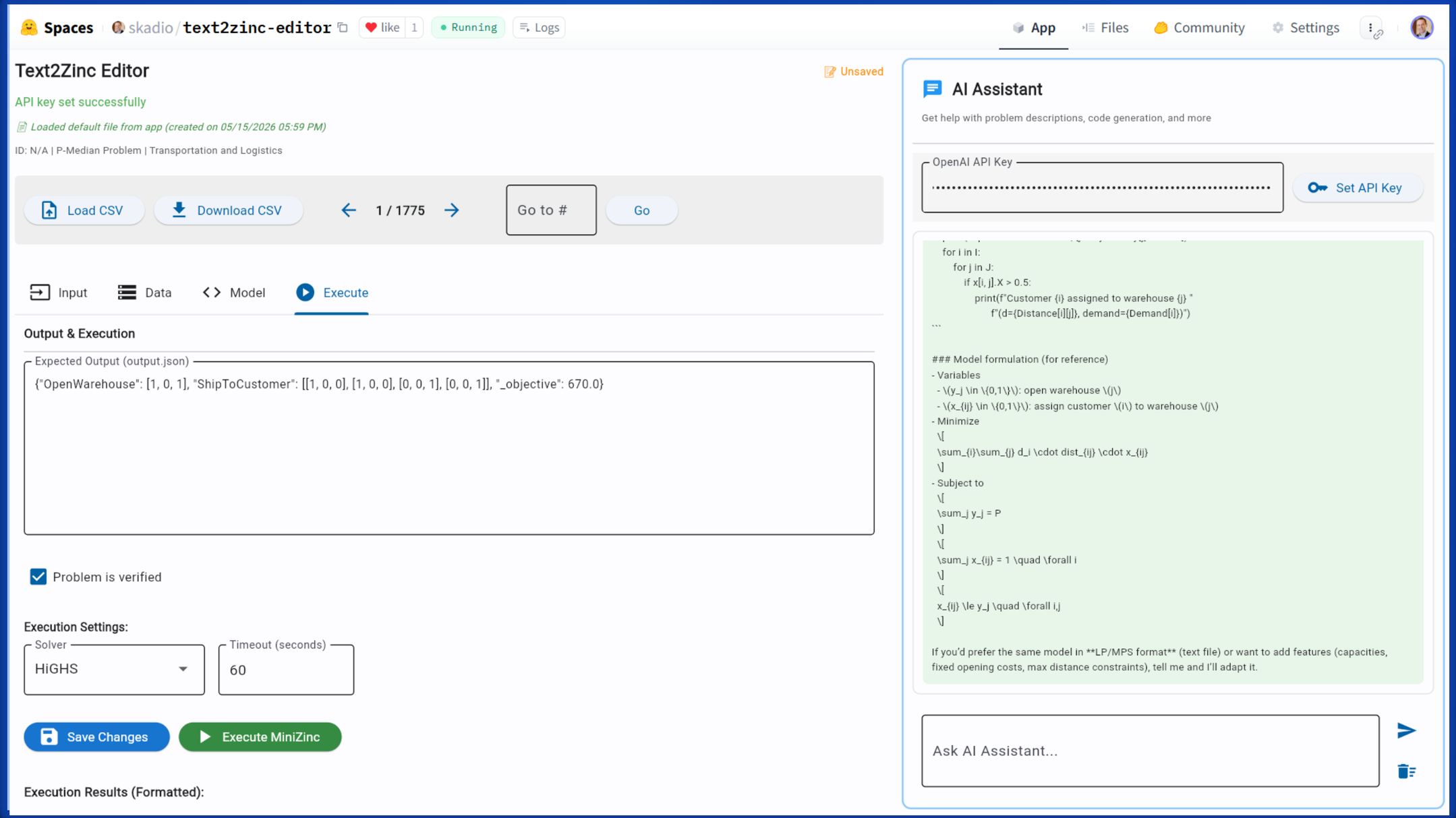
Text2Zinc Interactive Editor

https://huggingface.co/spaces/skadio/text2zinc-editor

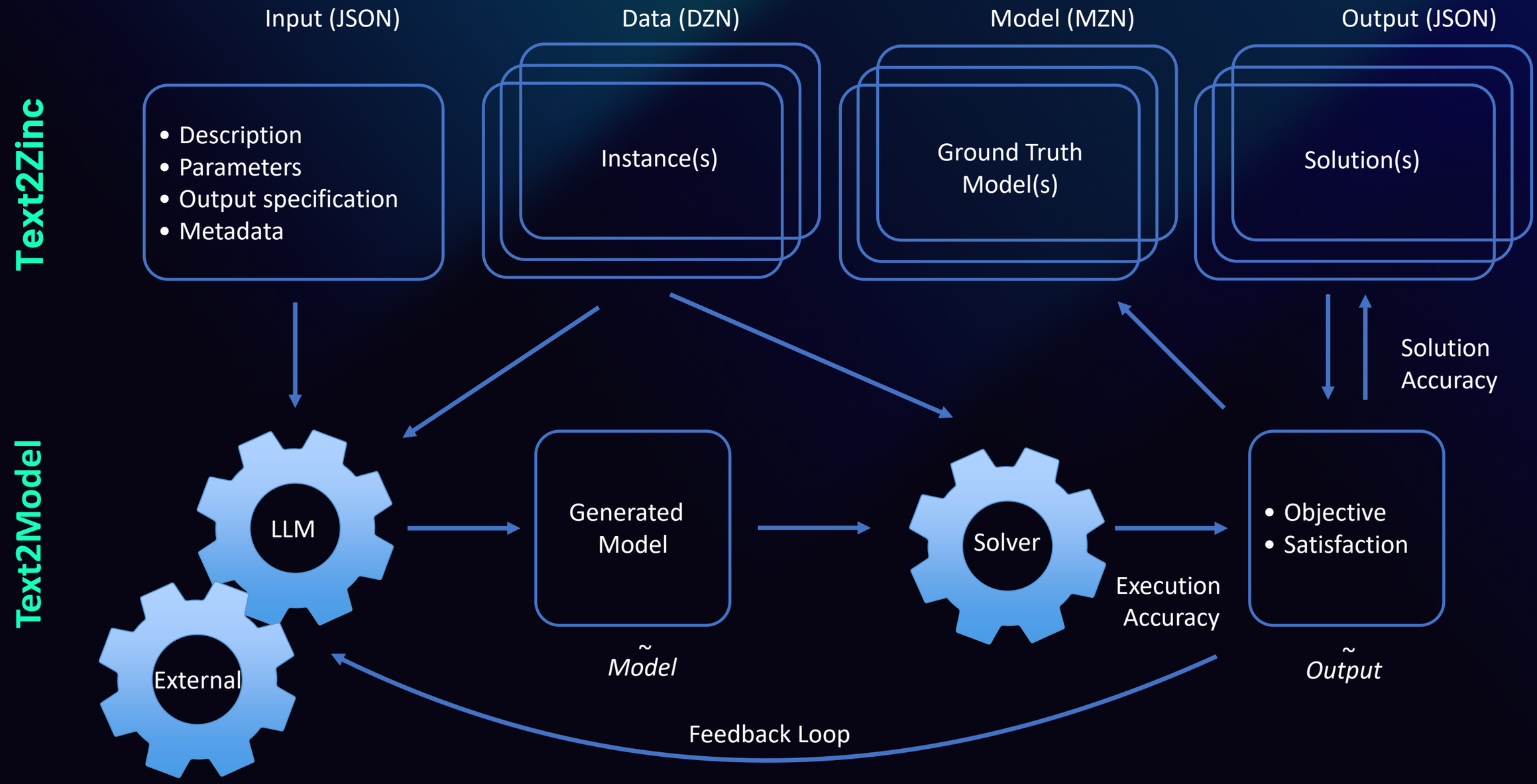


Text2Zinc Interactive Editor

https://huggingface.co/spaces/skadio/text2zinc-editor



Text2Zinc Dataset & Text2Model Copilots



Text-to-Model Translation

Text2Zinc Dataset

A **unified cross-domain dataset** curated to work with **LLM co-pilots** and **interactive editor** *Singirikonda et. al., AAAI'25*

Text2Model Copilots

A suite of **LLM Modeling copilots** with multiple strategies of varying complexity and **leaderboard** *Kadioglu et. al*

Gala: Global LLM Agents

A **global agentic approach** with specialized agents decompose model global constraint *Cai et. al. NeurIPS'25*

Learn2Zinc

Targeted **fine-tuning** to teach small language models to generate syntactically correct MiniZinc *Kadioglu et. Al*

Ner4Opt

Extracting optimization components from free-form natural language text. *Kadioglu et. al, Constraints'24*

Holy Grail 2.0

Blueprint for **optimization modelling co-pilots** with feedback loop and user interactivity *Tsouros et. al., 2023*

Text2Model Co-Pilots

Out-of-the-box LLM

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

Structured Prediction

Grammar-based model generation to enforce LLM output

Chain-of-Thought & Agentic

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

Knowledge Graph

Leveraging structured knowledge as intermediary representation

Knowledge Graph Generation

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

01

Identify Parameters

Extract each parameter with type, name, bounds, and alignment to predefined nomenclature.

02

Identify Variables

Note types, names, and bounds; cross-verify with problem description for completeness.

03

Identify Constraints

Detail each constraint with description, formula, associated variables, and classification (linear, nonlinear, global).

04

Identify Objective

Define objective function, optimization type (min/max/satisfy), and relevant constraints.

05

Generate TTL Representation

Construct **Turtle-format (TTL) KG** sequentially organizing parameters, variables, constraints, and objective.

Grammar Validation

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

→ Problem Context

Problem description and data nomenclature for semantic grounding.

→ Current Code + Error

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

→ MiniZinc Grammar Spec

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

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

MiniZinc Grammar Example

Grammar-constrained generation relies on formal grammar rules to validate generated code.

Consider this **simple rule for Boolean literals:**

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

What This Rule Enforces

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

Why It Matters

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

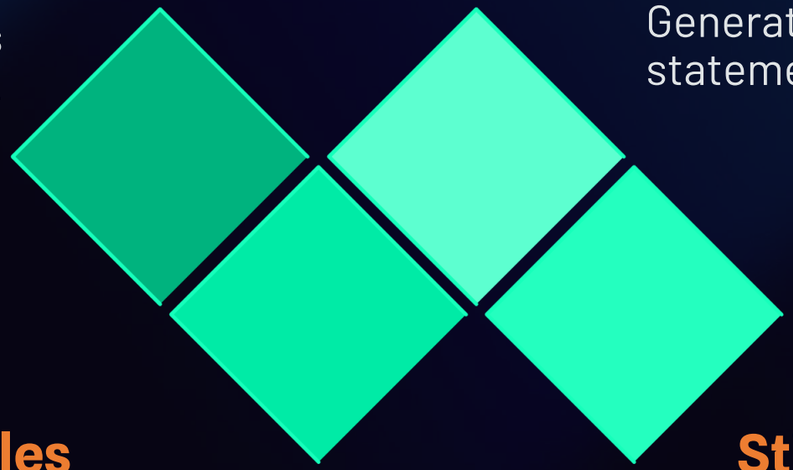
Agentic Decomposition

Constraints

Generate constraint blocks ensuring global correctness.

Objective

Generate the solve statement for optimization.



Parameters & Variables

Declare all inputs and decision variables.

Stitch

Combine sections into a complete MiniZinc model.

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

Benchmark Results: GPT-5.2 (Part I)

Strategy	LLM Calls	NLP4LP E/S	CSPLIB E/S	HAKANK E/S
Zero-Shot	1	63.08 / 32.31	63.64 / 36.36	50.00 / 31.82
Chain-of-Thought	1	70.77 / 35.38	54.55 / 27.27	59.09 / 36.36
Knowledge Graph	2	52.31 / 32.31	36.36 / 27.27	68.18 / 31.82
CoT + Grammar	2	**76.92 / 37.26**	**90.91 / 45.23**	**72.73 / 37.9**
Agentic + Code	5	76.92 / 29.23	81.82 / 36.36	68.18 / 36.36

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

Benchmark Results: GPT-5.2 (Part II)

Strategy	LLM Calls	INDUSTRYOR E/S	MAMO-EASY E/S	MAMO-COMP E/S	NL4OPT E/S
Zero-Shot	1	66.00 / 46.00	95.71 / 79.75	48.82 / 27.96	93.88 / 74.29
Chain-of-Thought	1	80.00 / 49.00	98.62 / 83.90	57.35 / 39.34	**98.78 / 78.37**
CoT + Grammar	2	**92.00 / 56.00**	**99.85 / 83.74**	**95.26 / 54.03**	97.96 / 77.96
Agentic + Code	5	86.00 / **57.00**	99.23 / **84.97**	92.89 / 54.03	97.55 / 77.96

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

General Observations

1 Execution-Solution Gap

Consistently lower solution accuracies across strategies indicate **complexity of modeling** expertise even for frontier models

2 Compilation Issues

Syntax errors are primary cause of execution failures, attributed to LLM's limited exposure to **MiniZinc's specialized syntax**

3 Information Sweet Spot

Both too little and too much information can be detrimental, suggesting an optimal **level of context** exists

4 Reasoning vs. Structure

Superior **performance of CoT and structured** approaches suggests how information is processed matters more than quantity provided

Frontier Model + CoT + Grammar Encoding leads but..

1271

CoT + Grammar Exec acc

Best execution accuracy

956

CoT + Grammar Sol acc

Best solution accuracy

362

Unsolved Instances

27% remain beyond current LLM capabilities

27% of instances remain out of reach

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

LLM Copilots for Text-to-Model Translation

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

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

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

Paper | Copilots | Slides | Leaderboard | Interactive Editor | Collections

https://skadio.github.io/text2model

What’s Next?

1. **Call-to-Action: Dataset & Copilot Expansion**

   Encourage community contributions to create more comprehensive resources

2. **Context Engineering**

   Explore alternative intermediate representations and semantic graphs

3. **Agentic SLM Frameworks**

   Investigate agentic approaches in capturing nuances of modeling problems

4. **Specialized SLMs**

   Develop specialized SLMs for optimization and satisfaction tasks

Text-to-Model Translation

Text2Zinc Dataset

A **unified cross-domain dataset** curated to work with **LLM co-pilots** and **interactive editor** *Singirikonda et. al., AAAI'25*

Text2Model Copilots

A suite of **LLM Modeling copilots** with multiple strategies of varying complexity and **leaderboard** *Kadioglu et. al*

Gala: Global LLM Agents

A **global agentic approach** with specialized agents decompose modeling by global constraint. *Cai et. al. NeurIPS'25*

Learn2Zinc

Targeted **fine-tuning** to teach small language models to generate syntactically correct MiniZinc *Kadioglu et. Al*

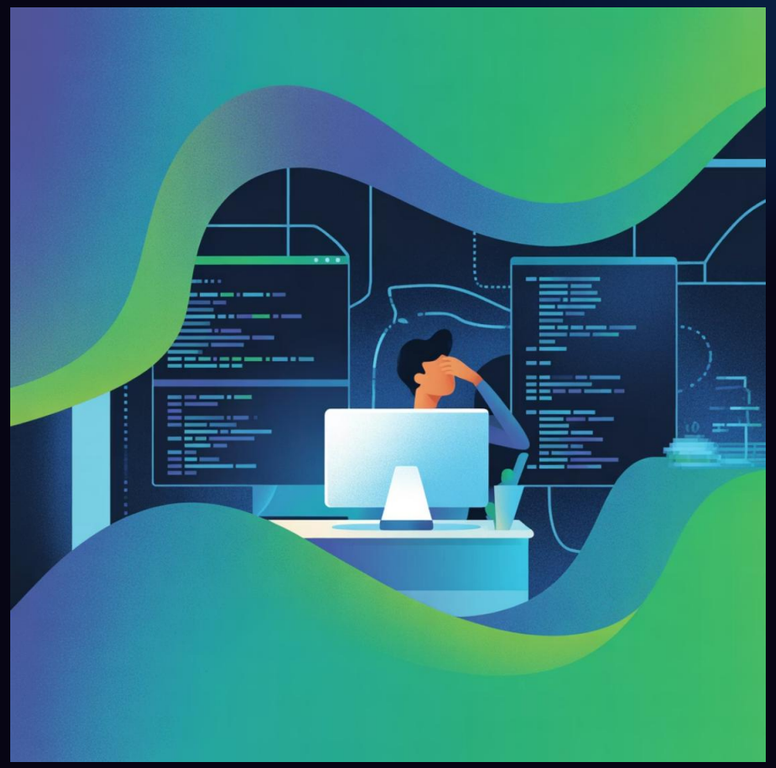
Ner4Opt

Extracting optimization components from free-form natural language text. *Kadioglu et. al, Constraints'24*

Holy Grail 2.0

Blueprint for **optimization modelling co-pilots** with feedback loop and user interactivity *Tsouros et. al., 2023*

The Challenge: Precise Logic & Domain Knowledge



Current State

Notoriously difficult to translate into correct MiniZinc models. This process demands both **logical reasoning** and **modeling expertise.**

Problem

General-purpose prompting often **fails to capture** all variables and constraints correctly, especially for harder optimization problems.

Current approaches using and even powerful LLMs are **"not yet a push-button technology"** for generating combinatorial models from text.

Motivation

This motivates research into more structured and guided methods that **break problems into manageable pieces.**

Multi-Agents for LLM Co-Pilots

Multi-step and multi-agent frameworks have emerged as promising solutions for natural language optimization tasks. By dividing complex problems into manageable pieces, each LLM handles a **simpler reasoning challenge,** potentially reducing overall complexity.

01

Chain-of-Experts

Assigns **multiple LLM experts** with specific roles (interpreting text, formulating components, coding, verifying) coordinated by a conductor Xiao et al. [2023]

02

OptiMUS System

Uses **LLM-based agent** to iteratively identify parameters, write constraints, and debug linear program models AhmadiTeshnizi et al. [2023]

03

Promising Results

These approaches considerably **improve over single-LLM methods** on complex operations research problems.

Breaking Down the Complexity Further

Current Limitations

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

Our Novelty

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

Agents meets Constraint Programming

all_different

Enforces distinct values across variables

cumulative

Enforces scheduling resource capacity over time

global_cardinality

Limits how many variables take each value

circuit

Creates a Hamiltonian circuit

Key Advantage

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

Global Constraints

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

Gala Agents

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

Agentic Gala Framework

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

Decomposition

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

Specialized Agents

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

Integration

Assembler agent integrates constraint snippets into complete model.

How Agents Work?

Specialized LLM Agents

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

Binary Classification

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

No Broader Modeling

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

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

The Assembler: Bringing It All Together

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

01

Declare Variables

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

02

Analyze Constraints

Decide whether to include provided global constraints

03

Fill Gaps

Add remaining constraints from text not covered by hints

04

Define Objective

Determine optimization objective or satisfy goal

05

Finalize Model

Append solver boiler plate and output format

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

Initial Evaluation Framework

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

1

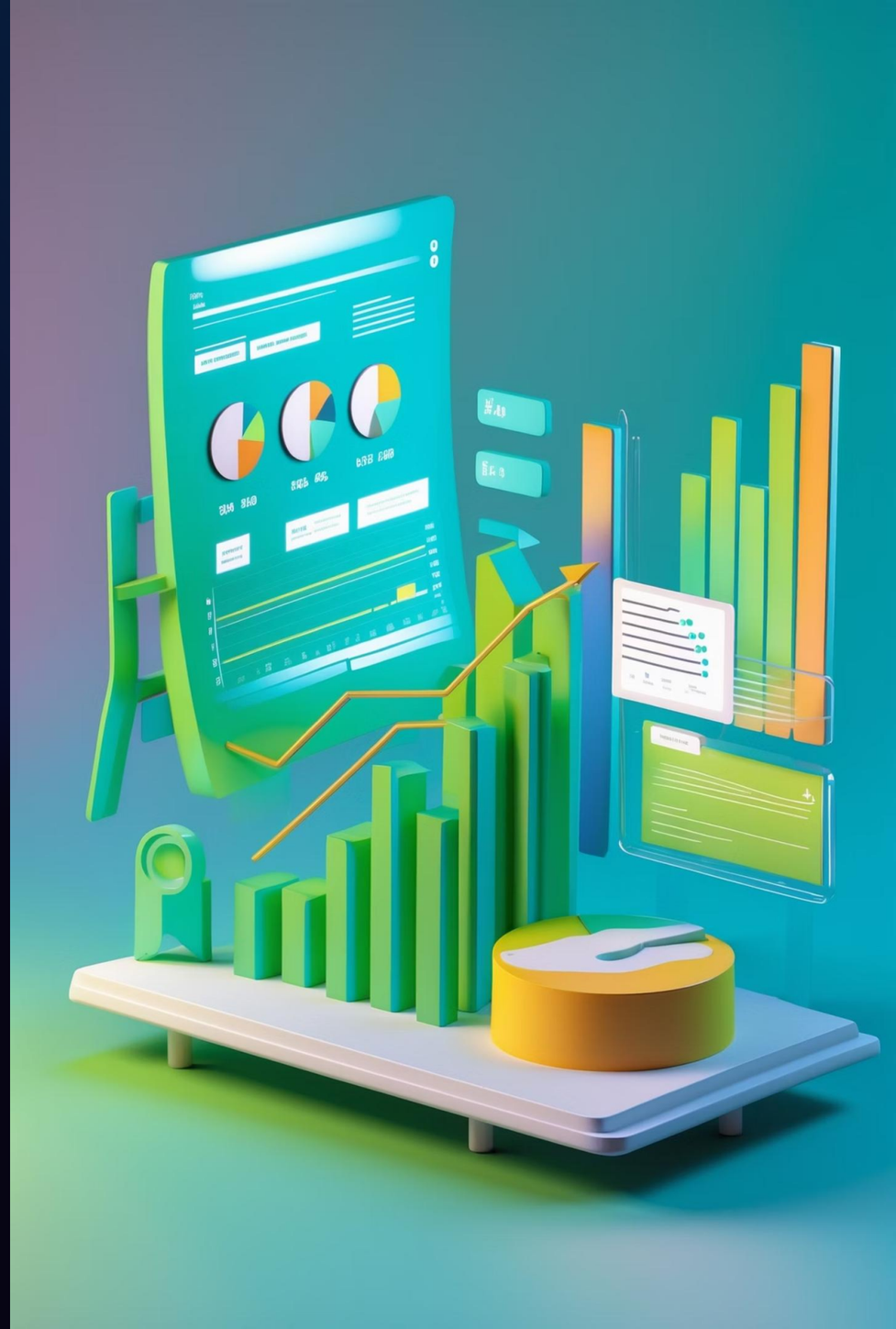
Global Agent Detection

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

2

End-to-End Performance

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



Global Agent Detection Performance

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

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

Strong Detection Rates

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

Room for Improvement

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

Gala End-to-End Modeling Performance

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

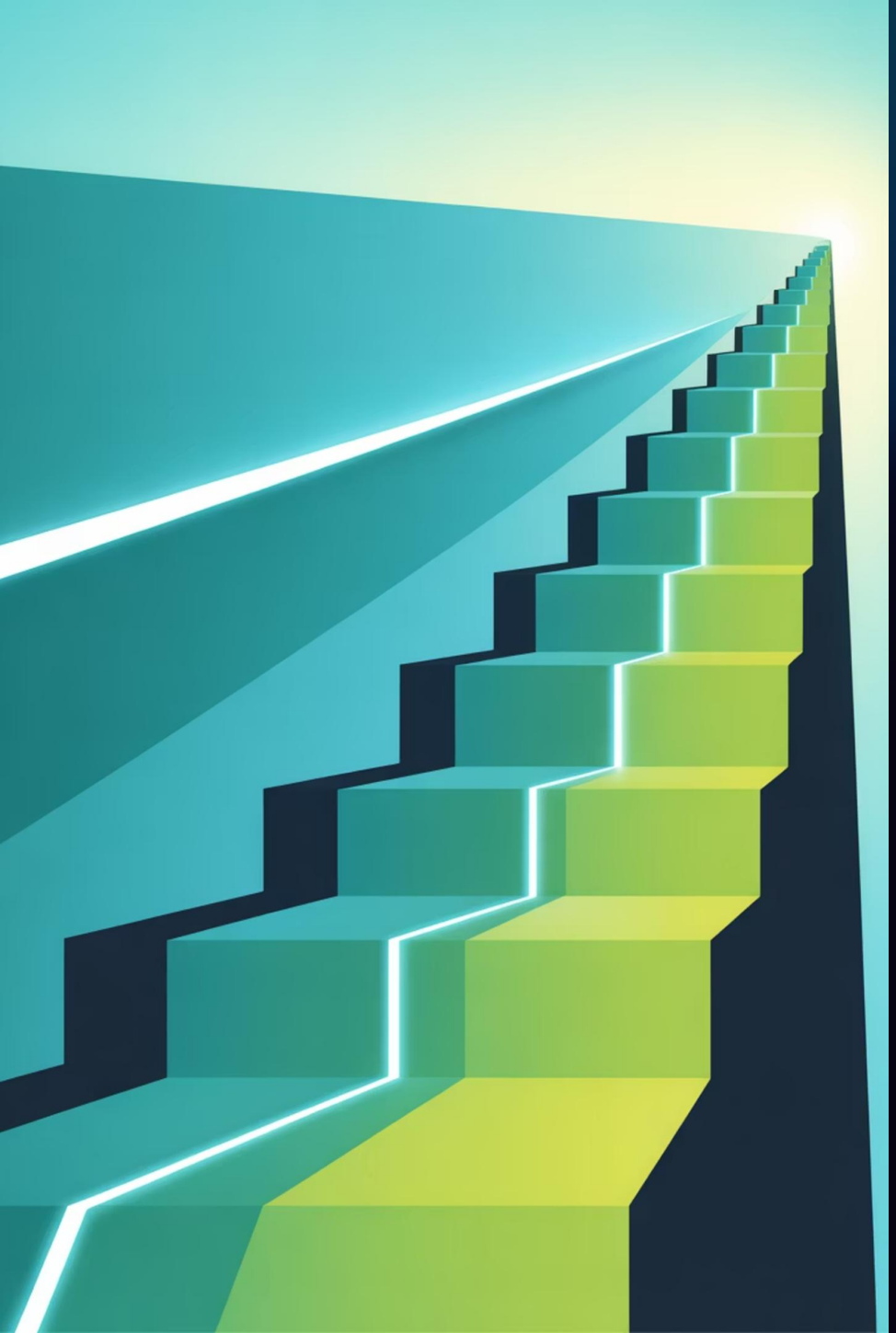
Model & Strategy	Execution Rate (%)	Solve Rate (%)
o3-mini Gala	**57.27**	**32.73**
o3-mini CoT	52.73	30.91
gpt-4o-mini Gala	**33.64**	**17.27**
gpt-4o-mini CoT	31.82	12.73
gpt-oss:20b Gala	**17.27**	8.18
gpt-oss:20b CoT	16.36	**10**
gpt-oss:20b baseline	11.81	7.27

Consistent Performance

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

Decomposition Advantage

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



Potential Enhancements

Optimize Global Agents

Replace hand-crafted prompts with **systematic optimization**, **curated few-shot** exemplars, and adversarial negatives. Consider **fine-tuning per global constraint** to boost precision and recall.

Unblock the Assembler

Add supervisor to **extract variables and objectives** before delegation. Build systematic error taxonomy to map where agents succeed or fail, driving targeted fixes and **feedback loops**.

Scale Evaluation

Run **stronger LLMs** (e.g., GPT-5) and sweep both open- and closed-weight models. Benchmark on **global constraint rich datasets** to better showcase the approach.

MLxOR@NeurIPS'25

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

https://skadio.github.io/text2model

Text-to-Model Translation

Text2Zinc Dataset

A **unified cross-domain dataset** curated to work with **LLM co-pilots** and **interactive editor** *Singirikonda et. al., AAAI'25*

Text2Model Copilots

A suite of **LLM Modeling copilots** with multiple strategies of varying complexity and **leaderboard** *Kadioglu et. al*

Gala: Global LLM Agents

A **global agentic approach** with specialized agents decompose model global constraint *Cai et. al. NeurIPS'25*

Learn2Zinc

Targeted **fine-tuning** to teach small language models to generate syntactically correct MiniZinc *Kadioglu et. al*

Ner4Opt

Extracting optimization components from free-form natural language text. *Kadioglu et. al, Constraints'24*

Holy Grail 2.0

Blueprint for **optimization modelling co-pilots** with feedback loop and user interactivity *Tsouros et. al., 2023*

MiniZinc is Out-of-Distribution for SLMs

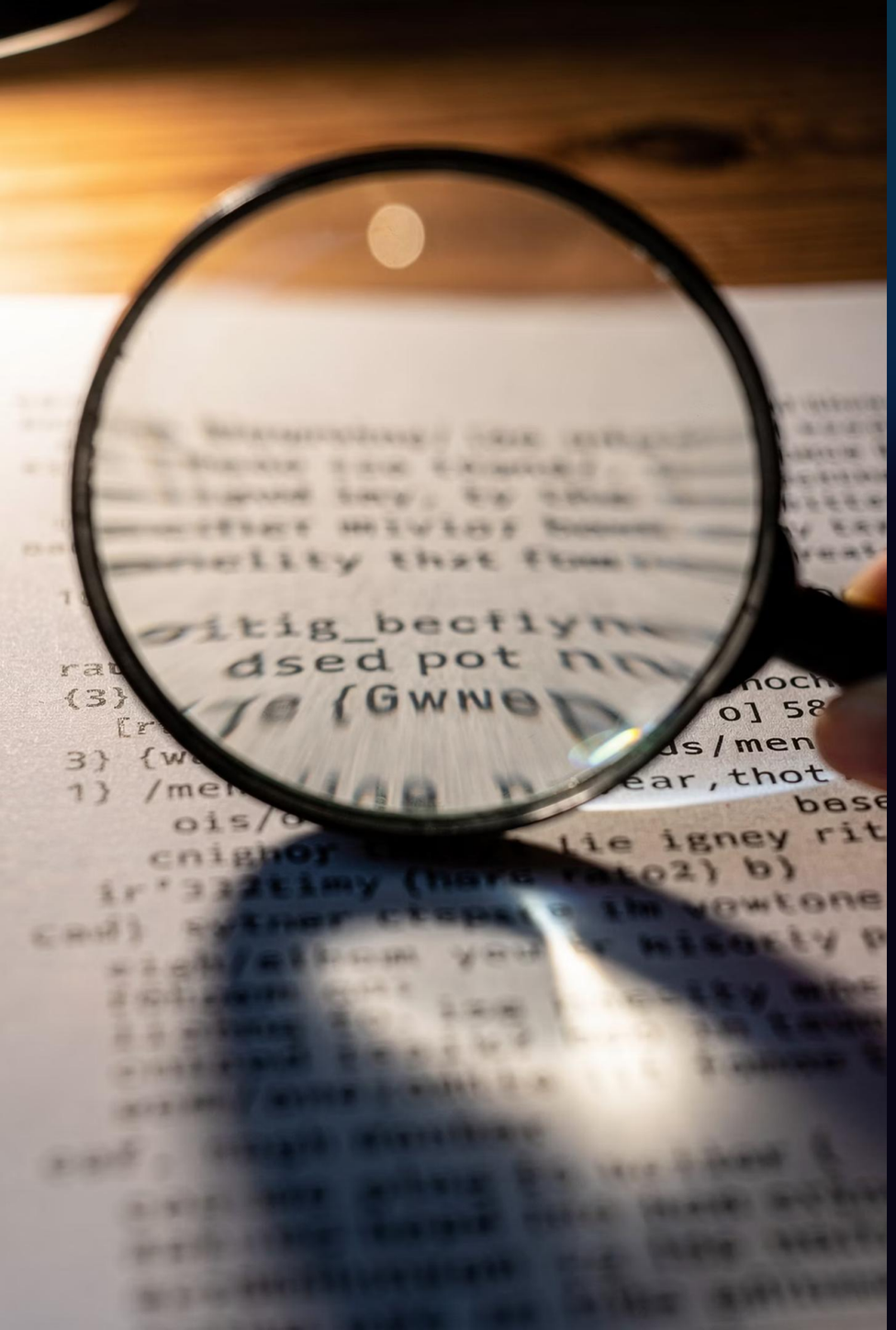
Complete Failure Without Fine-Tuning

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

0%

Qwen3, LLaMa3, Gemma2

Execution accuracy out-of-the-box



The Core Insight: Verification Enables Fine-Tuning

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

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

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

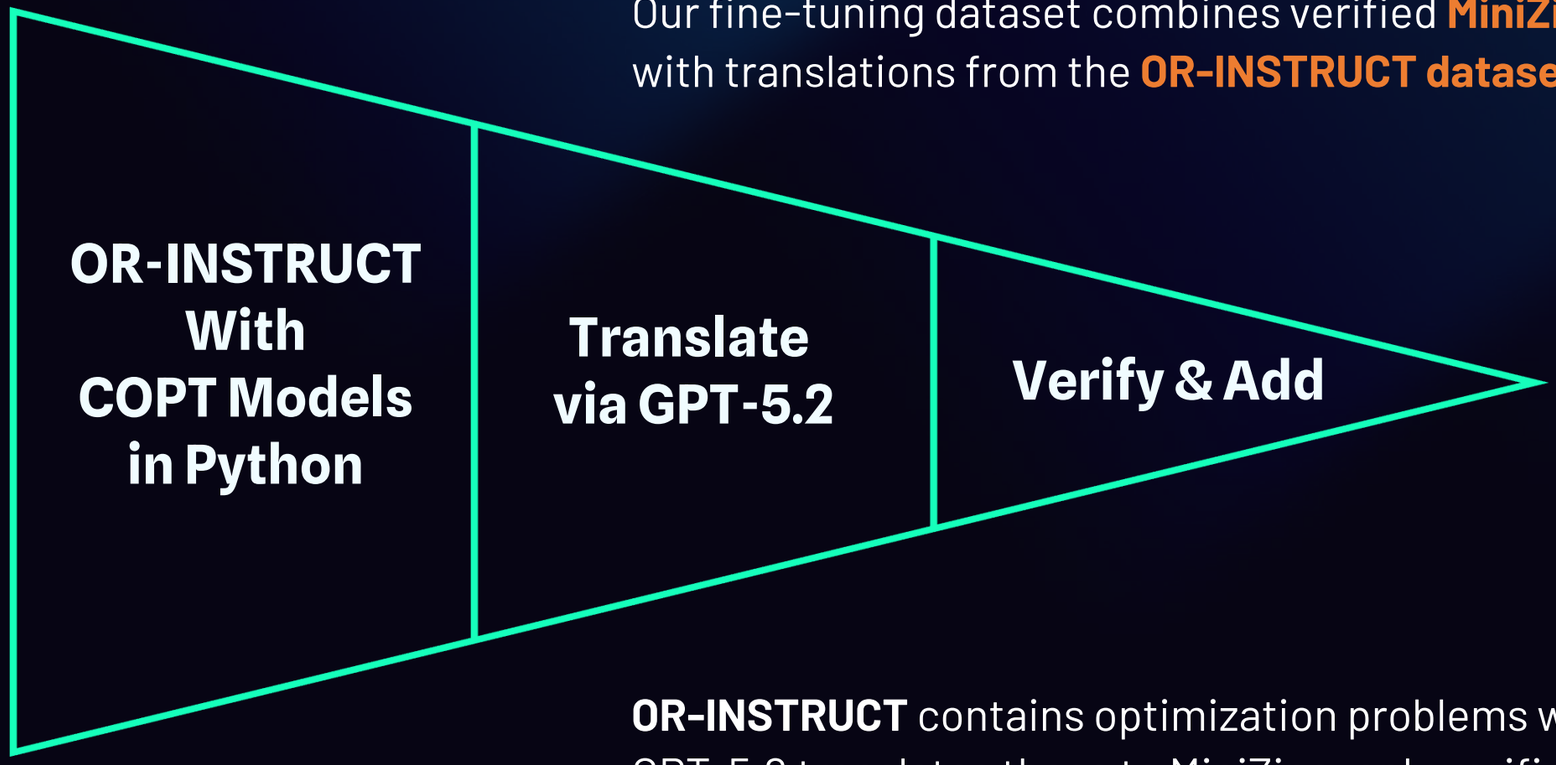
This yields a large pool of **verified** **(text, model)** **pairs** that makes fine-tuning possible *without requiring manual annotation.*

First Systematic Study

Fine-tuning SLMs for MiniZinc code generation.

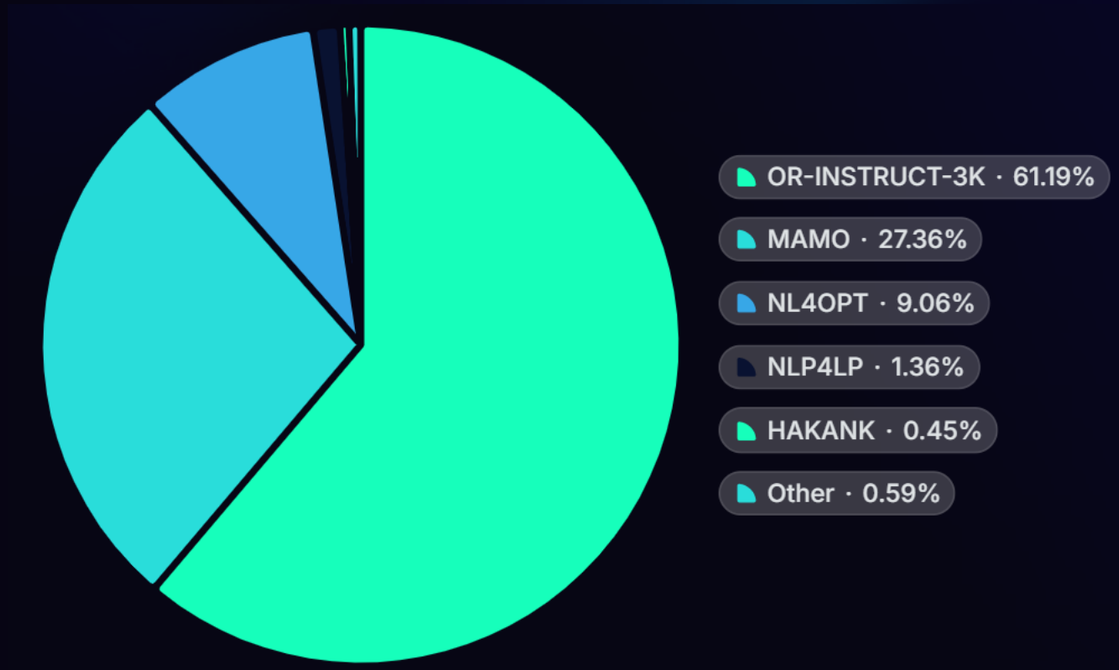
Building the Fine-Tuning Dataset

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



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

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



Summary

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

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

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

OR-INSTRUCT dominates at **61.2%**, with MAMO contributing **27.4%**.

Fine-Tuning Methodology

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

1) Generation Fine-Tuning

Teach SLMs to produce valid MiniZinc from natural language descriptions.

2) Error-Correction Fine-Tuning

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

Small Language Models

Qwen3-0.6B

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

LLaMA-3.2-1B & 3B

Mid-range models from Meta's LLaMA family.

Gemma-2-9B

Google's 9B parameter model.

GPT-OSS-20B

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

Three Fine-Tuning Datasets

Learn2Zinc-Base

8,014 pairs of
(text, model)

Direct **code generation** training.

Learn2Zinc-CoT

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

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

Learn2Zinc-Base+CoT

16,028 pairs
combining both Base and CoT

Mixed
training strategy.

Training Setup: LoRA on a Single GPU

Why LoRA?

We employ **Low-Rank Adaptation (LoRA)** with 8-bit quantization (4-bit for GPT-OSS-20B) to **fit all models on a single A100 GPU**.
Full fine-tuning would **overfit** on our ~8K instance dataset and is computationally **infeasible** at 20B scale.

LoRA Advantages

- Prevents overfitting on small datasets
- Enables 0.6B–20B fine-tuning on a **single GPU**
- Well-established PEFT approach with strong task adaptation
- Balances training efficiency with performance gains

Five SLMs × three datasets = **15 fine-tuning experiments**.

Benchmark: Text2Zinc-IndustryOR

100 Text2Zinc-IndustryOR instances as test set

the most challenging benchmark as reported across multiple studies.

86%

Best Exec Accuracy

CoT + Grammar GPT-5.2

(Kadioğlu et al. 2026)

57%

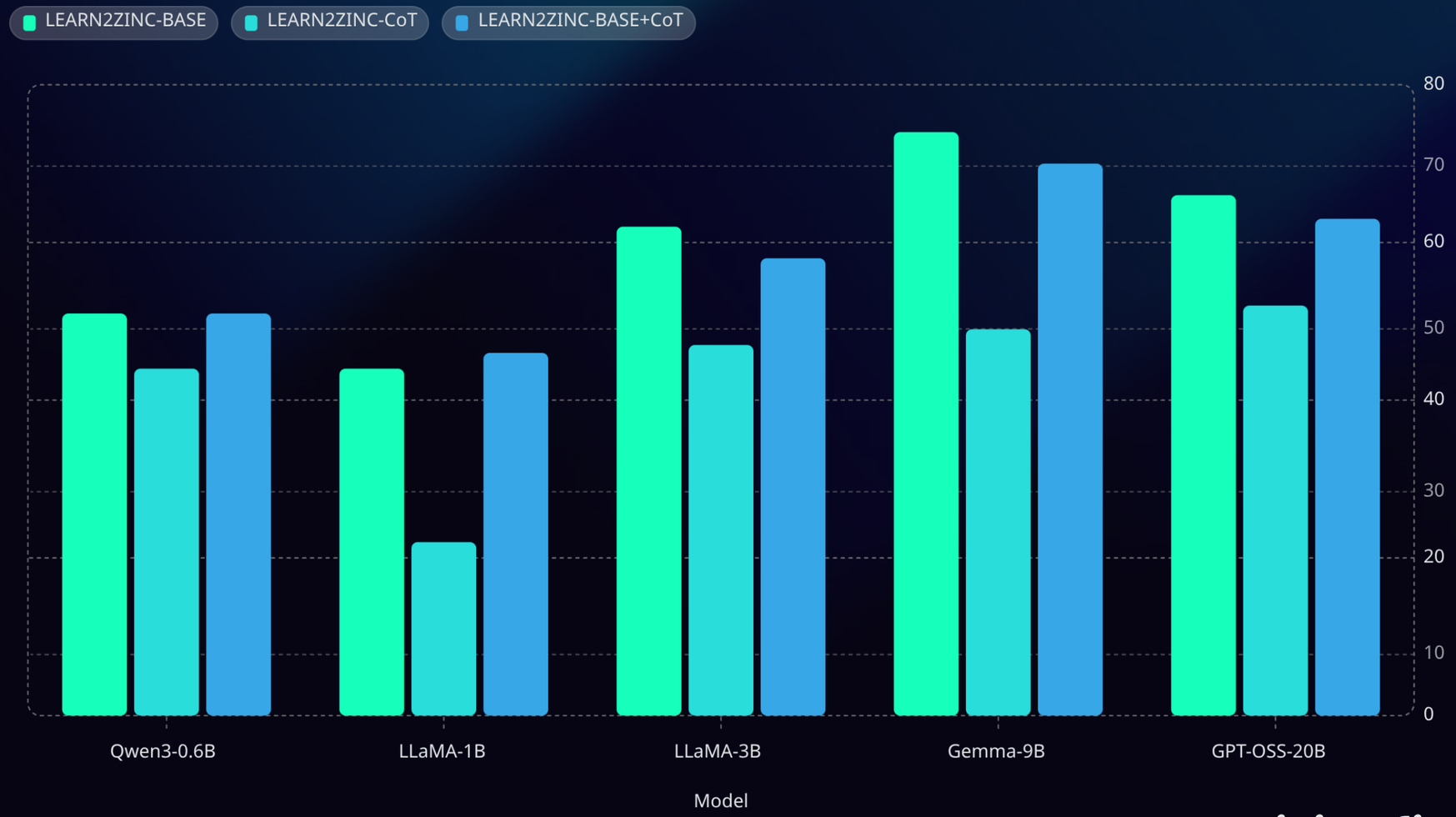
Best Sol Accuracy

CoT + Grammar GPT-5.2

(Kadioğlu et al. 2026)

NOT included in fine-tuning

Numerical Results: Execution Accuracy



Learn2Zinc-CoT **consistently underperforms** all other strategies — a ***surprising finding*** that contrasts with prompting frontier models.

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

Near-Zero Baseline

Remarkable Boost

SLMs vs. Frontier (86%-57%)

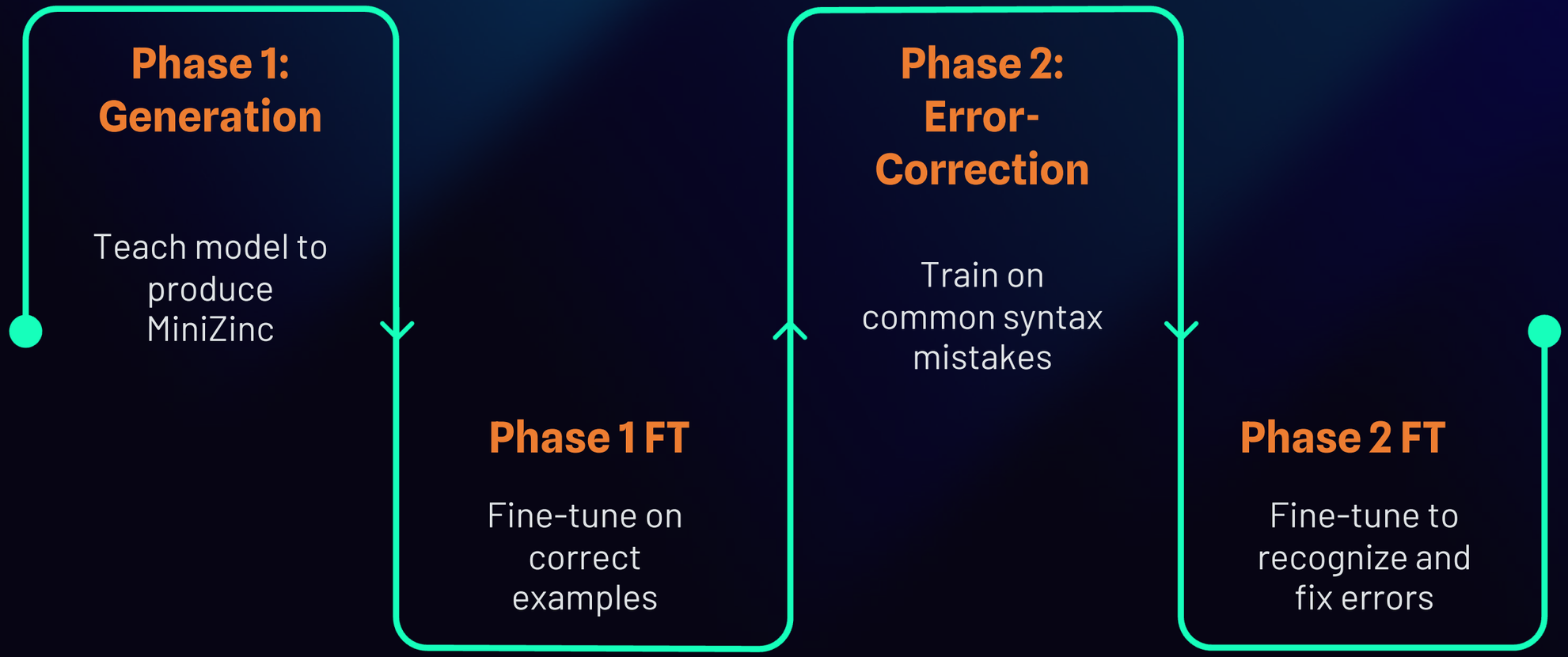
TEACHING SYNTAX

The Syntax Bottleneck

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

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

Shifting to Error-Correction Fine-Tuning



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

Error Correction Dataset: Synthetic Corruptions

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

Easy (7 rules)

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

Medium (6 rules)

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

Hard (7 rules)

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

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

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

Taxonomy of Corruption Rules

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

Cross-Model Error Bootstrapping

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

01

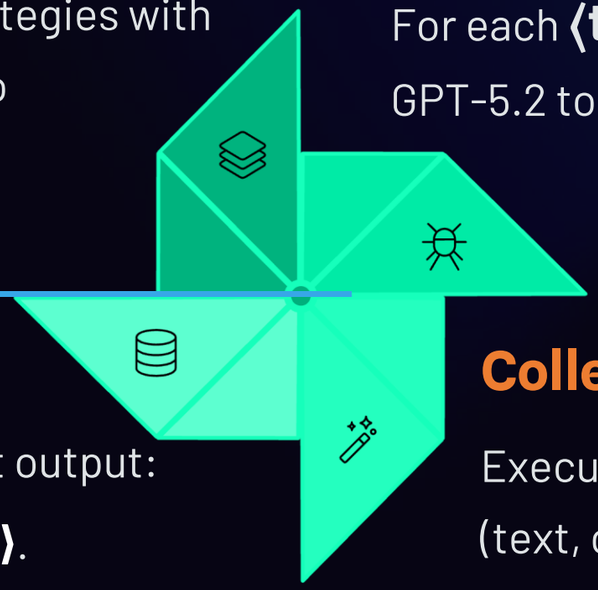
Run All 15 SLMs

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

02

Collect Execution Failures

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



03

Verify & Add to Dataset

If the fix compiles and produces correct output: add **⟨text, corrupt_model, model⟩**.

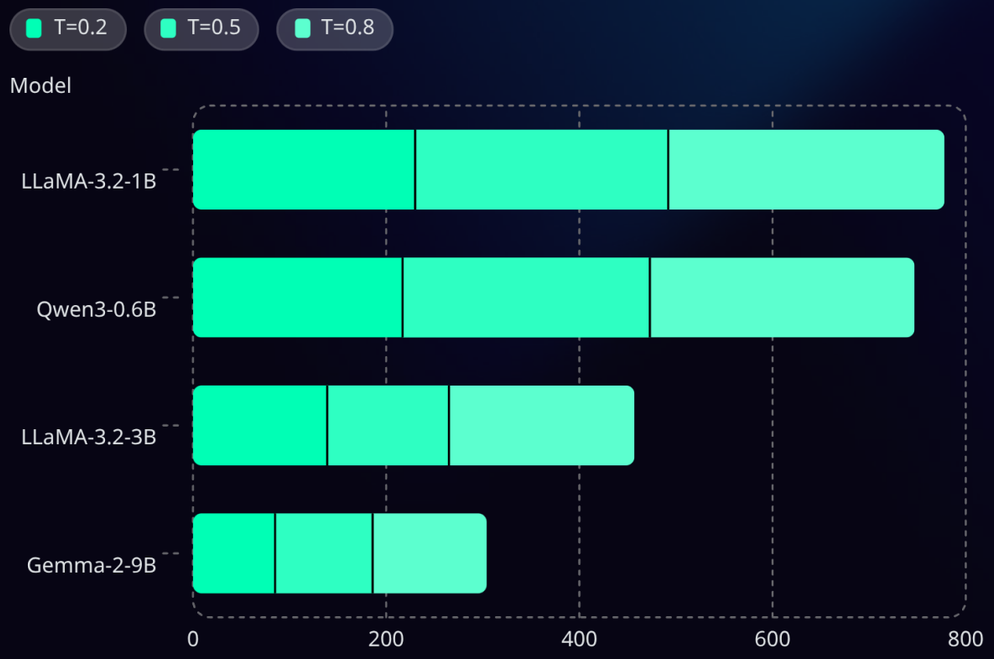
Yields **2,286 verified correction instances**.

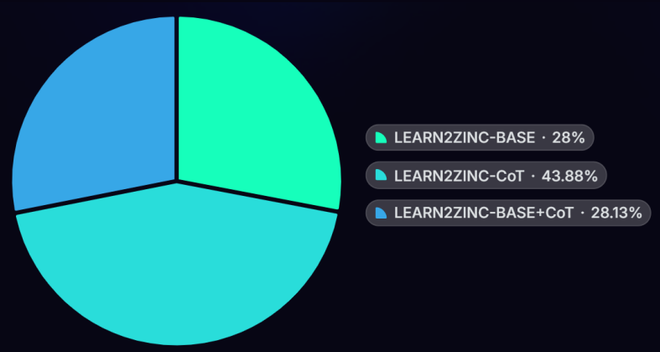
04

Collect Successes Too

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

Error Distribution by Model & Temperature





Smaller models contribute the most correction examples. An interesting property: **small models learn from large model mistakes, and vice versa.**

Higher sampling temperatures produce more errors temperature 0.8 accounts for **38.1%** of corrections vs. 29.4% at temperature 0.2.

CoT produces the most errors interesting as our surprise finding that SLM cannot benefit reasoning traces.

Augmented Dataset with Dual Fine-Tuning Objective

4,452

Synthetic Corruptions

Grammar-based error-correction pairs

2,286

Cross-Model Corrections

Real LLM failure corrections via GPT-5.2

8,911

Generation Examples

Verified (text, model) pairs for code generation

15,649

Total Instances

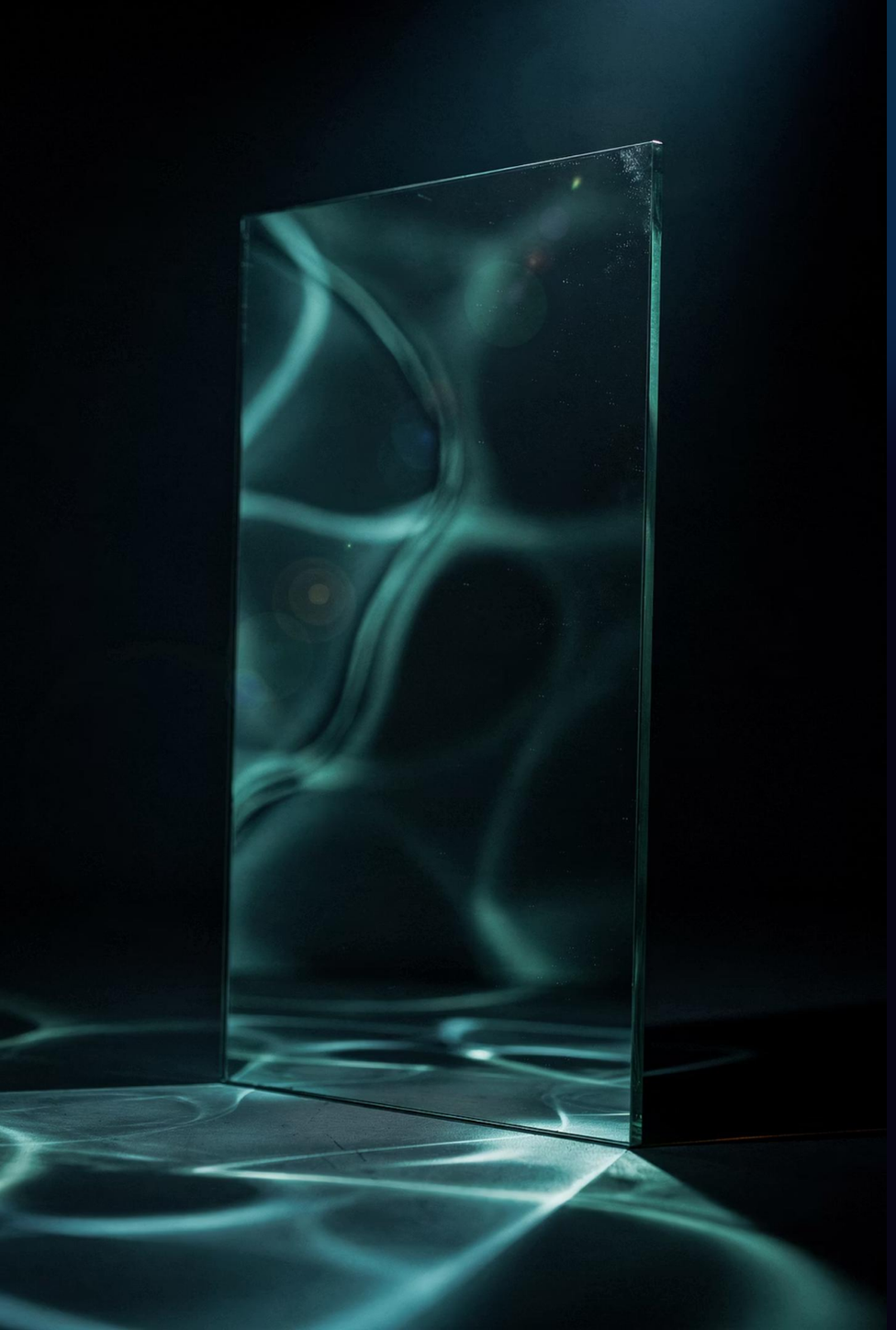
Combined augmented fine-tuning dataset

Learn2Zinc Augmented

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

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

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



Beyond Zero-Shot: Self-Reflection & Ensemble

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

Self-Reflection Loop

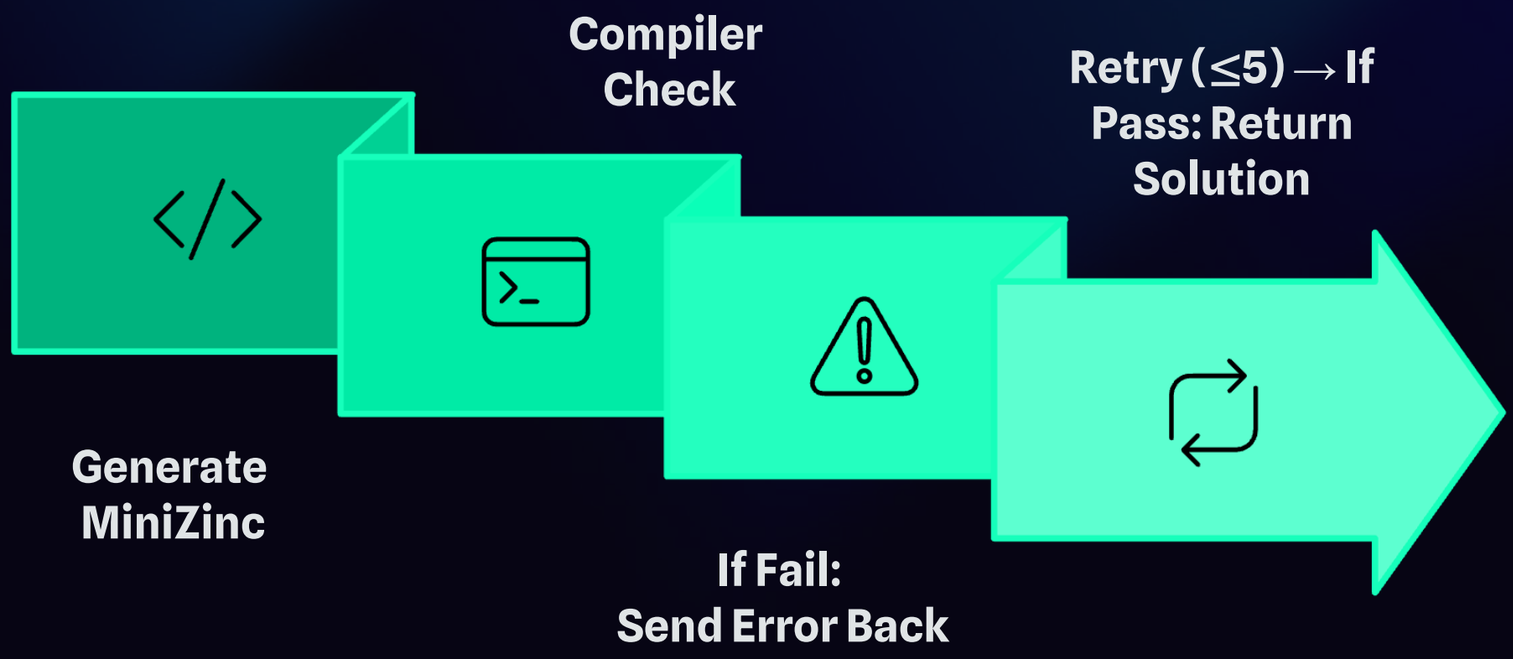
Model receives its broken code + compiler error message and retries — up to 5 attempts.

Top-Down Ensemble

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

Self-Reflection up to 5 Attempts

On each failed attempt, the model receives its broken code and the compiler error message, then tries again



Reflections reach a running model **in less than two trials on average**, with larger SLMs requiring fewer attempts.

Self-Reflection Results

Model	Single Exec (%)	Single Sol (%)	Reflection Exec (%)	Reflection Sol (%)	Avg Tries
Qwen3-0.6B-Aug	64.0	13.0	72.0	13.0	2.22
LLaMa-3.2-1B-Aug	57.0	8.0	71.0	8.0	2.35
LLaMa-3.2-3B-Aug	70.0	17.0	80.0	17.0	1.90
Gemma-2-9B-Aug	72.0	22.0	84.0	24.0	1.79
GPT-OSS-20B-Aug	76.0	32.0	**89.0**	**34.0**	**1.62**

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

89%

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

Our Augmented SLM

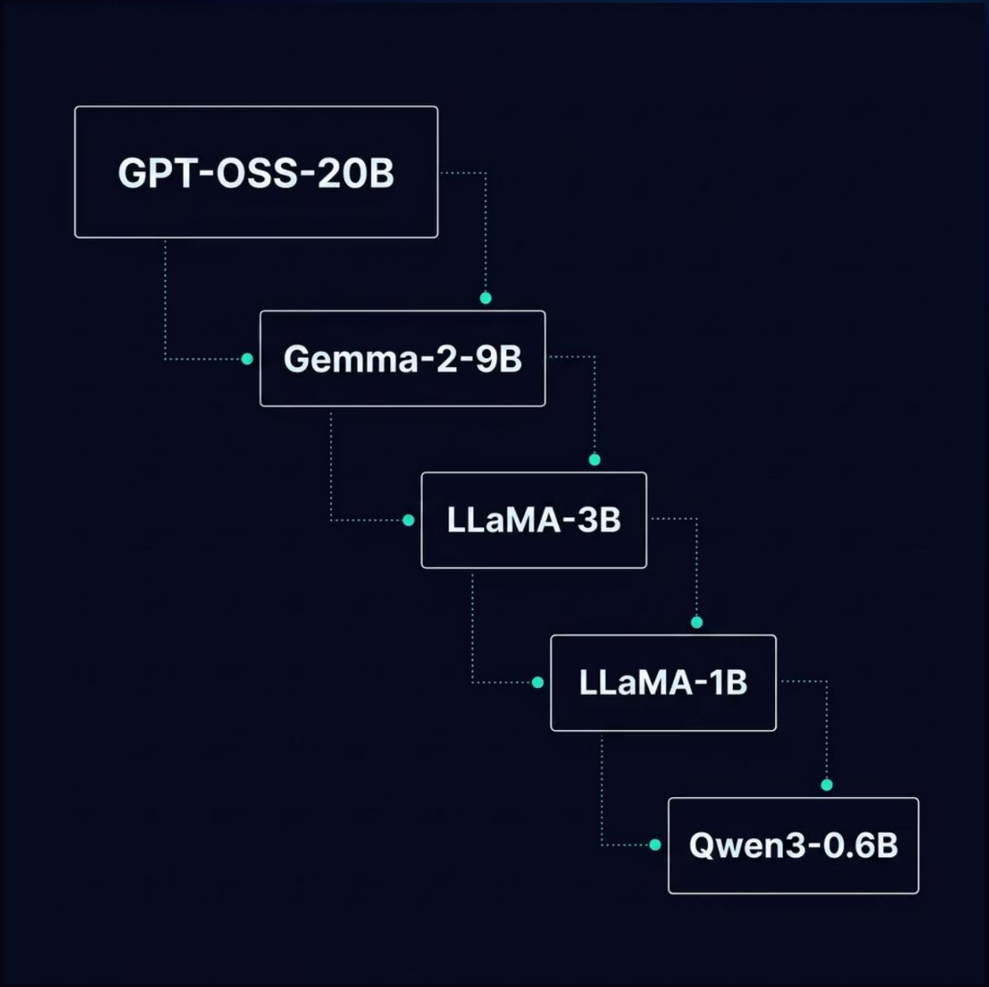
GPT-OSS-20B: **6%** → **76%** → **89%** with self-reflection. Achieves on-par execution accuracy with its frontier variant.

Frontier GPT-5.2

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

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

Top-Down Ensemble Strategy



Why Top-Down?

Models are tried in **descending order of capability**. If a model fails to produce executable code after retries, the next model is attempted.

Top-down maximizes the chance of getting an executable model where **solution accuracy remains highest** (stronger models first).

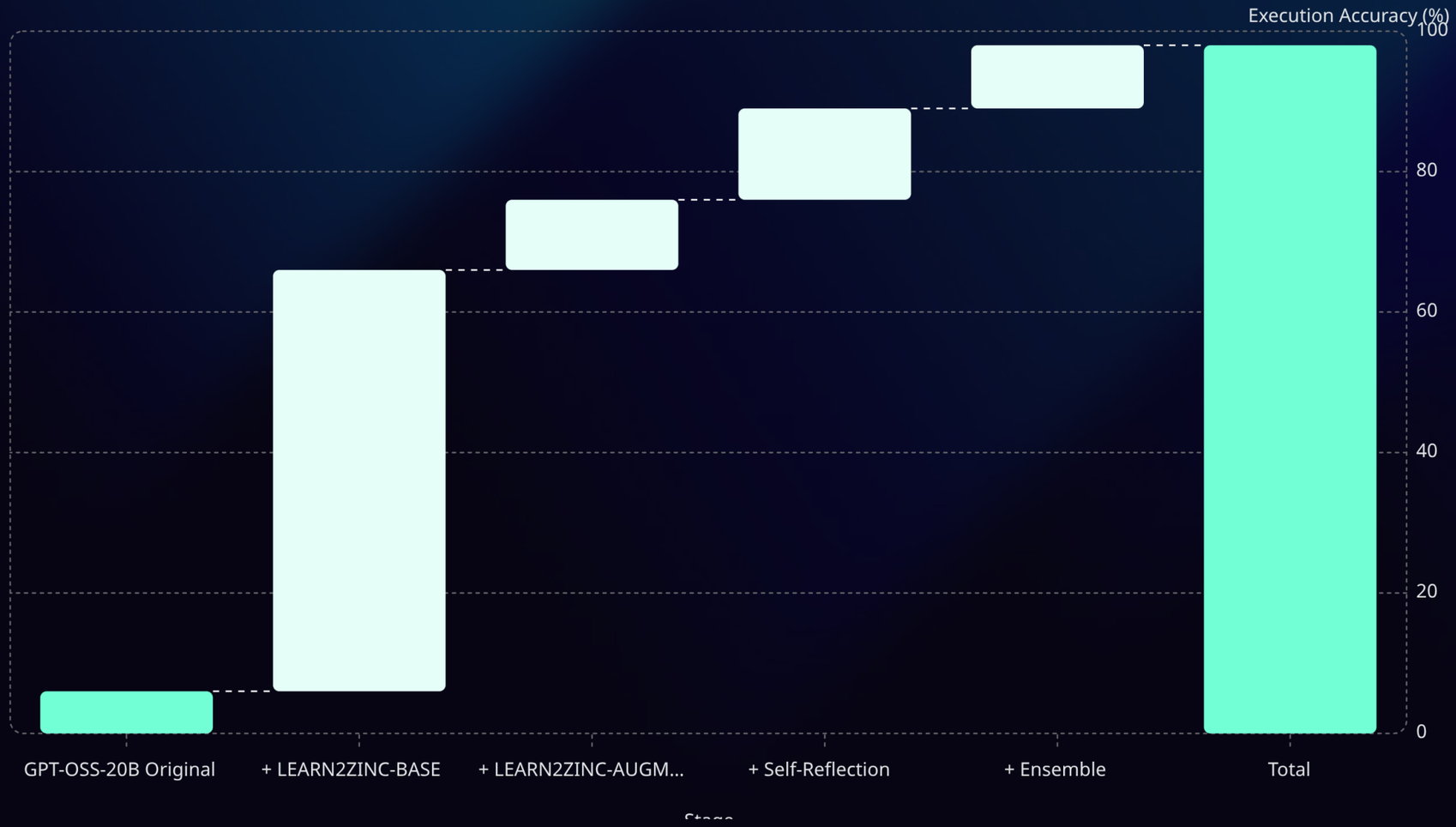
Bottom-up would risk obtaining an executing model when reasoning capacity is lowest.

98% Execution Accuracy

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

SLMs complete failure without fine-tuning

Learn2Zinc: Overall Progression



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

Semantic Error Analysis

Type I: Contradictory Constraints

Models **introduce auxiliary variables** with two incompatible definitions — e.g., a variable constrained to equal both leftover supply and truck count from the same quantity. Solver returns UNSAT even though the problem is feasible.

Type II: Phantom Variables

Models create **unnecessary decision variables** (e.g., binary indicators in a pure LP) constrained to equal expressions exceeding their declared bounds. The only satisfying assignment sets all variables to zero — giving a trivially wrong objective.

Type I errors alone could improve solution accuracy by up to **18% points**.

Type II errors alone could improve solution accuracy by up to **20% points**.

Directional Comparison on IndustryOR

Method	Model	Calls	IndustryOR Sol (%)
ORLM - Pass@1(Huang et. al. 2024)	LLaMA-3-8B	1	38.0
ORLM - Pass@8	LLaMA-3-8B	8	49.0
Lean-LLM-Opt (Liang et. al. 2024)	GPT-OSS-20B	5+	59.0
Lean-LLM-Opt	GPT-4.1	5+	**65.0**
Text2Model: Cot + Grammar	GPT-5.2	5	**57.0**
Learn2Zinc-Augmented	GPT-OSS-20B	1	32.0
Learn2Zinc-Augmented+Reflection	GPT-OSS-20B	1.62	**34.0**

Learn2Zinc with GPT-OSS-20B is below **Pythonic** methods, reflecting both MiniZinc's syntax challenge and SLM capacity.
Text2Model with GPT-5.2 shows stronger models with **MiniZinc** can approach Pythonic performance.
ORLM tied to COPT and Lean-LLM-Opt tied to Gurobi and are only applicable to optimization.

Code Generation for Rare Languages

EsoLang-Bench Finding

Not only SLMs but even frontier models drop from **85–95% to 0–11%** on rare programming languages.

Few-shot prompting and self-reflection provide negligible benefit when pretraining coverage is absent.

Closest Related Work: OptiMind

OptiMind (Zhang et al. 2026)

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

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

Call for Future Research

Fine-tuning models for one domain-specific language may **degrade capabilities in others**. This raises important questions about:

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

Learn2Zinc: Key Take Aways

Novelty

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

Result

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

Insight

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

→ **Distillation Reasoning from Larger Models**

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

FINE-TUNING

Learn2Zinc: Teaching SLMs MiniZinc

Fine-tuning open-weight small-language models (SLMs) of various sizes from 0.6B to 20B parameters to generate MiniZinc models from text. We share our fine-tuning pipeline, datasets, models.

Serdar Kadıoğlu, Karthik Uppuluri

Fine-tuned Models

Qwen3	0.6B	Llama 3.2	1B	Llama 3.2	3B
Gemma 2	9B	GPT-OSS	20B		

arXiv'26 | Code | Fine-tuning Dataset | Fine-tuned Models | BibTeX

https://skadio.github.io/text2model

Strategic Pillars of Enterprise AI @ Fidelity AI Center

AI Learning from Offline Data

Robust, scalable, reproducible features from structured, unstructured, and semi-structured datasets.

Selective, TextWiser, Seq2Pat

AI for Learning from Online Feedback

Adaptive, real-time, A/B testing systems that continuously learn from user interaction.

Mab2Rec, MABWiser

AI for Decision Making

Large-scale, integrated, (meta) solvers for resource management and optimization.

Forge, Balans, PathFinder

AI for Automated Assistants

Extraction and translation of natural language into downstream tasks and intents for human-computer interaction.

Text2Model, Text2Zinc, Learn2Zinc, Gala, Ner4Opt, iCBS

Responsible AI

Horizontal capabilities for explainability, evaluation, fairness, and bias mitigation across all systems.

Jurity, BoolXAI, BoolXLLM

Open-Source AI at Scale: Establishing an Enterprise AI Strategy [AI Magazine'25]