

This is the DEDICATION:
you can write whatever you want here,
or nothing at all . . .

Introduction

This is the introduction

Contents

Introduction	i
1 Technical Background	1
1.1 Constraint Programming	1
1.1.1 CP Solvers	2
1.1.2 MiniZinc	2
2 Experimental Evaluation	5
2.1 Methodology	5
2.1.1 Provider Choice	5
2.1.2 Large Language Models Selection	7
2.1.3 Prompt General Structure	7
2.1.4 Problem Selection	8
2.1.5 Test Metrics	9
2.1.6 Experiment Setup	12
2.2 Experiments	14
2.2.1 Preliminary tests	14
2.2.2 Single Request Experiments	15
2.2.3 Multi-turn Experiments	17
2.2.4 Feature Employment	20
2.2.5 Sampling Temperature Tuning	22
Appendix	25
Bibliography	37

Chapter 1

Technical Background

In this chapter, the main intention is to give the reader an initial knowledge base on all the theoretical concepts and all the components that had been utilized in the development of this research.

... spiegazione per sezione ...

1.1 Constraint Programming

Constraint programming (CP) is an emergent software technology for declarative description and effective solving of large, particularly combinatorial, problems especially in areas of planning and scheduling [1].

A constraint can be thought of intuitively as a formalization of dependencies in physical worlds and their mathematical abstractions. A constraint is a logical relation among several unknowns (or variables), each taking a value in a given domain. The constraint thus restricts the possible values that variables can take; it represents partial information about the variables of interest. Constraints can also be heterogeneous, so they can bind unknowns from different domains, for example a length (number) with a word (string). The important feature of constraints is their declarative manner, i.e., they specify what relationship must hold without specifying a computational procedure to enforce that relationship [1].

Constraints arise naturally in most areas of human endeavor. They are the natural medium of expression for formalizing regularities that underlie the computational and (natural or designed) physical worlds and their mathematical abstractions.

Constraint programming is the study of computational systems based on constraints. The idea of constraint programming is to solve problems by stating constraints (requirements) about the problem area and, consequently, finding a solution satisfying all the constraints.

As stated by E. C. Freuder thirty years ago, “Constraint Programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it.” [2].

1.1.1 CP Solvers

In the courses of this research paper, we will often talk about “Solvers”. A CP solver is defined as a system or program that manipulates constraints to find solutions that satisfy specified conditions. It can utilize various approaches to enhance flexibility and customization within specific constraint domains, particularly finite domains.

So, given that a CP model is a tuple $\langle X, D, C, O \rangle$ where X is the set of variables we are trying to assign a value to, $D(x)$ is the domain associated with each variable, C is the set of constraints that the variables must respect, and O is an objective function, the goal of the solver is to assign a value for each variable $x \in X$ from $D(x)$ that satisfies all the constraints in C and optimizes the objective function O [4].

Portfolio Solvers

Solving combinatorial search problems is hard, and there exist nowadays plenty of techniques and constraint solvers for performing this task. It has become clear that different solvers are better when solving different problem instances, even within the same problem class. It has also been shown that a single, arbitrarily efficient solver can be significantly outperformed by using a portfolio of possibly on-average slower solvers [5].

Algorithm portfolios [6] can be seen as instances of the more general Algorithm Selection problem [7] where, as reported in [8], the algorithm selection is performed case-by-case for each problem to solve. Within the context of constraint solving, a portfolio approach enables to combine a number $m > 1$ of different constituent solvers s_1, \dots, s_m in order to create a globally better constraint solver, dubbed a portfolio solver. When a new, unseen problem p comes, the portfolio solver tries to predict the best constituent solver(s) s_{i_1}, \dots, s_{i_k} (with $1 \leq i_j \leq m$ for $j = 1, \dots, k$) for solving p and then runs them on p . Properly selecting and scheduling the solvers is a crucial step for the performance of a portfolio solver, and it is usually performed by exploiting Machine Learning techniques based on features extracted from the problem p to solve.

In the course of this research paper, starting from the idea behind portfolio solvers, we will try to evaluate the performances of an agentic solver that uses pre-trained large language model as its agent to understand if this could be a viable solution to the algorithm selection problem [7], following the new paradigm proposed in [9].

1.1.2 MiniZinc

MiniZinc is a simple but expressive CP modelling language which is suitable for modelling problems for a range of solvers and a reasonable compromise between many design possibilities.

Born from the necessity of a standard modelling language for constraint programming problems also making solver benchmarking simpler [3].

For these reasons all of the problems employed for testing in the course of this research paper were written using MiniZinc.

Specifying a Problem

A MiniZinc problem specification has two parts: (a) the model, which describes the structure of a class of problems; and (b) the data, which specifies one particular problem within this class. The pairing of a model with a particular data set is a model instance (sometimes abbreviated to instance).

The model and data may be in separate files. Data files can only contain assignments to parameters declared in the model. A user specifies data files on the command line, rather than naming them in the model file, so that the model file is not tied to any particular data file [3].

MiniZinc Challenge

Comparing constraint programming systems is fraught with difficulty and in reality an impossible task. The reason is that there are so many components to a modern CP system, only some of which are implemented by some systems. To claim that one CP system is “better” than another is a bold claim, since there is almost certainly some problem for which the “worse” system allows a stronger model, or a better search, and performs better.

Chapter 2

Experimental Evaluation

2.1 Methodology

In this section, the focus is on outlining the foundational decisions required to establish an initial benchmark. This benchmark serves as the basis for refining subsequent experiments and assessing both the current capabilities and future potential of the solver.

The section is structured into four subsections, each addressing a key preliminary choice. The first subsection discusses the selection of large language models used as candidates for the agentic component. The second details the initial prompt-engineering strategy needed to define a clean, consistent prompt format for evaluation. The third presents the rationale behind the selection of benchmark problems used to test model performance. The fifth explains the metrics adopted to evaluate how effectively each model could operate as a meta-solver. The final subsection explains the pre-processing methods adopted to make automatic testing possible.

2.1.1 Provider Choice

To build the proposed Agentic Solver (AS), the first requirement is the availability of an LLM capable of orchestrating the system and acting as the agent. Given the limited computational resources available during the testing phase, we had to rely on externally hosted LLMs accessed through usage-based APIs. The selection prioritized generous free tiers, permissive rate limits, and straightforward integration. This led to the choice of the following providers:

- **Gemini API v1** [10], offered by Google DeepMind. Gemini is a family of large language models with multiple sizes and capabilities. This provider was selected for its strong reasoning abilities, robust tool-use features, and overall high-quality text generation. For the purpose of this research, the version v1 [11] was preferred as it is more stable and our only concern is its text generation capability.
- **Groq API** [12], provided by Groq. Groq offers high-performance inference solutions through its specialized hardware architecture. The Groq API exposes a selection of LLMs

through a simple and lightweight interface, enabling fast and low-latency experimentation.

Both APIs were selected for their ease of use, flexibility, overall performance, and, critically, their comparatively generous rate limits relative to competing services, in Table 2.2 and Table 2.1 are displayed rate limits of both APIs. From the leftmost column of the table, there are: Model containing the names of each one of the available LLMs (for the purpose of this paper, only text generation models were selected), moving to the right *RPM* contains the maximum number of requests in a minute, *RPD* contains the maximum number of requests per day, *TPM* contains the maximum number of requested tokens per minute, and finally *TPD* contains the maximum number of tokens per day.

Model	RPM	RPD	TPM	TPD
allam-2-7b	30	7000	6000	500000
deepseek-r1-distill-llama-70b	30	1000	6000	100000
gemma2-9b-it	30	14400	15000	500000
groq/compound	30	250	70000	–
groq/compound-mini	30	250	70000	–
llama-3.1-8b-instant	30	14400	6000	500000
llama-3.3-70b-versatile	30	1000	12000	100000
meta-llama/llama-4-maverick-17b-128e-instruct	30	1000	6000	500000
meta-llama/llama-4-scout-17b-16e-instruct	30	1000	30000	500000
meta-llama/llama-guard-4-12b	30	14400	15000	500000
meta-llama/llama-prompt-guard-2-22m	30	14400	15000	500000
meta-llama/llama-prompt-guard-2-86m	30	14400	15000	500000
moonshotai/kimi-k2-instruct	60	1000	10000	300000
moonshotai/kimi-k2-instruct-0905	60	1000	10000	300000
openai/gpt-oss-120b	30	1000	8000	200000
openai/gpt-oss-20b	30	1000	8000	200000
playai-tts	10	100	1200	3600
playai-tts-arabic	10	100	1200	3600
qwen/qwen3-32b	60	1000	6000	500000

Table 2.1: Rate limits - Groq models [13]:

This table shows all the offered models from Groq API, in leftmost column and each relative rate limit.

Model	RPM	RPD	TPM	TPD
gemini-2.5-pro	5	100	250000	–
gemini-2.5-flash	10	250	250000	–
gemini-2.5-flash-lite	15	1000	250000	–
gemini-2.0-flash	15	200	1000000	–
gemini-2.0-flash-lite	30	200	1000000	–

Table 2.2: Rate limits - Gemini models [14]

This table shows all the offered models from Gemini API, in leftmost column and each relative rate limit.

2.1.2 Large Language Models Selection

Both providers offer a broad set of LLMs with varying capabilities and constraints, so an initial filtering step was required. Several options were excluded immediately because they are not designed for text generation, which is essential for the proposed AS. In particular, `playai-tts` and `playai-tts-arabic` are text-to-speech LLMs available only on Groq’s platform and therefore unsuitable for remote testing.

Additional LLMs were removed because they are currently decommissioned or unavailable: `deepseek-r1-distill-llama-70b`, `gemini-2.0-flash-lite`, and `gemma2-9b-it`.

Two more LLMs were excluded due to insufficient context window size. Although their rate limits were acceptable, their token capacity was too small to accommodate even a single full MiniZinc model as input: `meta-llama/llama-prompt-guard-2-22m` and `meta-llama/llama-prompt-guard-2-86m`.

Finally, `allam-2-7b` was removed because it failed to follow instructions consistently, often producing incomplete, inconsistent, or unreadable outputs.

After this filtering stage, 18 LLMs remained as a stable base for the evaluation phase.

2.1.3 Prompt General Structure

To determine which LLM would be best suited for building an AS, it was necessary to design a consistent prompt format to query each model. The primary objective was to define a structure that was as short and clean as possible, for two main reasons:

- **Minimize prompt-induced bias:** A highly descriptive or too long and complex prompt could influence LLMs negatively. As we could encounter problems as “context rot” [15] - a progressive decay in accuracy as prompts grow longer.
- **Reduce token usage:** Since the testing setup depends on API limits, keeping the prompt compact minimizes token consumption.

Output Structure

Ensuring a standardized output format was equally important: Automated testing requires that model outputs follow a strict and predictable format. Any deviation introduces ambiguity during parsing and prevents reliable extraction of solver selections. Maintaining this structure is therefore essential to ensure consistent and fully automated evaluation.

Large or verbose responses also impose practical limitations on the available context window. Because each message contributes to the total token count, excessively long outputs reduce the room available for subsequent turns and larger prompts.

For these reasons, the output format was fixed as an array of three strings:

[“1stSolver“, “2ndSolver“, “3rdSolver“]

Selecting the top three solvers enables two forms of evaluation:

- Single-solver evaluation: Measures whether the solver chosen by the LLM is the single best solver for the given instance. If it is not, the evaluation can quantify how close its performance is to the optimal solver.
- Parallel-solver evaluation: Measures the effectiveness of running the top three solvers selected by the LLM in parallel. The best result among the three is considered, allowing assessment of whether any of them corresponds to the single best solver for the instance, or, if not, how close the best among the three comes to the optimal performance.

The metrics used for these evaluations will be detailed in subsection 2.1.5.

After all of this considerations, the resulting prompt structure is the one displayed in Figure 2.1

2.1.4 Problem Selection

A crucial component of the testing pipeline is the problem selection. Consistent and meaningful evaluation requires a set of benchmark problems that are reliable, diverse, and representative of real solver behavior. To meet these requirements, the problem set should satisfy the following criteria:

- Extensive prior testing: The problems must be validated and associated with reliable solver performance data, preferably obtained from recent evaluations of state-of-the-art solvers.
- Diversity: The set must include a varied mix of problem types-combinatorial problems, real-world applications, and puzzle-like tasks-covering all major categories: Maximization, Minimization and Satisfaction.

This ensures that LLM performance can be assessed across different solving paradigms.

Prompt Structure

MiniZinc model:

...Minizinc problem model (.mzn content) ...

MiniZinc data:

...Instance relative data (.dzn or .json content) ...

The goal is to determine which constraint programming solver would be best suited for this problem, considering the following options:

— s_1 ,

— s_2 ,

...

— s_n

where $s_{1...n} \in \text{SolverList}$ Answer only with the name of the 3 best solvers inside square brackets separated by comma and nothing else.

Figure 2.1: Example of prompt

- Complexity: The problems must be sufficiently challenging so that solver selection is non-trivial and the LLM’s reasoning abilities are meaningfully tested.

Following these criteria, the selected benchmark was the problem set from the *MiniZinc Challenge 2025* [18] [16] [17]. These problems are specifically curated to benchmark the strongest solvers of the year and therefore represent an ideal test bed for evaluating the proposed Agentic Solver.

The problem set contains twenty problems: 1 satisfaction problem, 3 maximization problems, 16 minimization problems.

Each problem is a combination of a `.mzn` file containing the Minizinc [3] model made of the high-level description of the problem (variables, constraints and objective function). Every problem also is also accompanied by five corresponding data instances each of them contained either in a `.dzn` or a `.json` file containing specific parameters and constants, yielding a total of 100 testable, diverse, and complex scenarios.

2.1.5 Test Metrics

In order to actually evaluate model performance, it is necessary to chose a standard metric for answer evaluation, other than that, it is necessary to have a metric to evaluate how an AS controlled by the given LLM would perform against the current Single Best Solver (SBS).

Before analysing the evaluation metrics, we must first define the systems to which these metrics will be applied. Namely, the solvers. In our context, a solver is a program that takes as input the description of a computational problem in a given language and returns an observable outcome providing zero or more solutions for the given problem. For example, for

decision problems, the outcome may be simply “yes“ or “no“ while for optimization problems, we might be interested in the best solutions found along the search. An evaluation metric, or performance metric, is a function mapping the outcome of a solver on a given instance to a number representing “how good“ the solver is on this instance. An evaluation metric is often not just defined by the output of the solver. Indeed, it can be influenced by other actors, such as the computational resources available, the problems on which we evaluate the solver, and the other solvers involved in the evaluation. For example, it is often unavoidable to set a **timeout** τ on the solver’s execution when there is no guarantee of termination in a reasonable amount of time (e.g. NP-hard problems). Timeouts make the evaluation feasible but inevitably couple the evaluation metric to the execution context. For this reason, the evaluation of a meta-solver should also consider the scenario that encompasses the solvers to evaluate, the instances used for the validation, and the timeout. Formally, at least for the purposes of this paper, we can define a scenario as a triple $(\mathcal{I}, \mathcal{S}, \tau)$, where: \mathcal{I} is a set of problem instances, \mathcal{S} is a set of individual solvers, $\tau \in (0, +\infty)$ is a timeout such that the outcome of solvers $s \in \mathcal{S}$ Solver instance $i \in \mathcal{I}$ is always measured in the time interval $[0, \tau]$. Evaluating meta-solvers over heterogeneous scenarios $(\mathcal{I}_1, \mathcal{S}_1, \tau_1)$, $(\mathcal{I}_2, \mathcal{S}_2, \tau_2)$, \dots , is complicated by the fact that the sets of instances \mathcal{I}_k , the sets of solvers \mathcal{S}_k and the timeouts τ_k can be very different. And things could get even more complicated in scenarios including optimization problems.

For those objectives two separate metrics were chosen

Metric for Solver Score

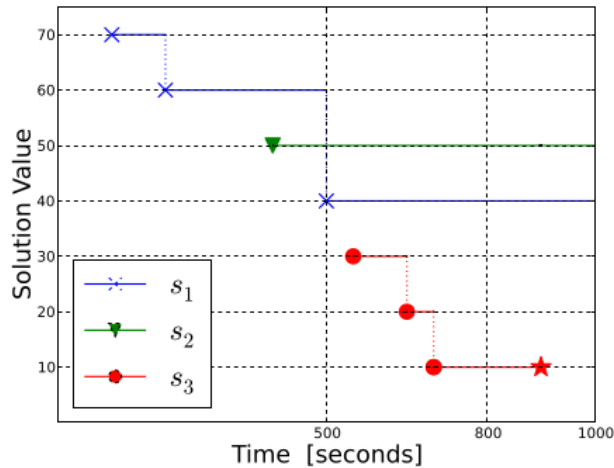


Figure 2.2: Solver performances example

We are now ready to associate to every instance i and solver s a weight that quantitatively represents how good is s when solving i over time T . We define the scoring value of s (shortly, score) on the instance i at a given time t as a function $\text{score}_{\alpha, \beta}$ [19] [20] defined as follows:

$$\text{score}_{\alpha,\beta}(s, i, t) = \begin{cases} 0, & \text{if } \text{sol}(s, i, t) = \text{unk}, \\ 1, & \text{if } \text{sol}(s, i, t) \in \{\text{opt}, \text{uns}\}, \\ \beta, & \text{if } \text{sol}(s, i, t) = \text{sat} \\ & \text{and } \text{MIN}(i) = \text{MAX}(i), \\ \max\left\{0, \beta - (\beta - \alpha) \frac{\text{val}(s, i, t) - \text{MIN}(i)}{\text{MAX}(i) - \text{MIN}(i)}\right\}, & \text{if } \text{sol}(s, i, t) = \text{sat} \\ & \text{and } i \text{ is a minimization problem}, \\ \max\left\{0, \alpha + (\beta - \alpha) \frac{\text{val}(s, i, t) - \text{MIN}(i)}{\text{MAX}(i) - \text{MIN}(i)}\right\}, & \text{if } \text{sol}(s, i, t) = \text{sat} \\ & \text{and } i \text{ is a maximization problem}. \end{cases}$$

Here, $\text{MIN}(i)$ and $\text{MAX}(i)$ denote the minimal and maximal objective function values found by any solver s at the time limit T .

As an example, consider the scenario in Figure 2.2 showing three different solvers on the same minimization problem. Let $T = 500$, $\alpha = 0.25$, $\beta = 0.75$. Solver s_1 finds the optimal value (40), therefore it receives score 0.75. Solver s_2 finds the maximal value (50), hence score 0.25. Solver s_3 does not find a solution in time, giving score 0. If instead $T = 800$, the value of s_1 becomes 0.375 and s_3 gets 0.75. If $T = 1000$, since s_3 improves the objective to 10 (marked with a star in the figure), it receives the highest score.

The parameter used for score calculation in testing are: $T = 1200000$ (1200000ms = 20 minutes, which is the time limit used solver evaluation in the MiniZinc Challenge) $\alpha = 0.25$ $\beta = 0.75$.

Closed Gap

Once the evaluation metric for solver score has been defined, we also need a comparative metric after score calculation. For this objective, we have chosen to use *closed-gap* [19] as the evaluation metric. Which is a relative and meta-solver-specific measure, adopted in the 2015 ICON and 2017 OASC [21] challenges to handle the disparate nature of the scenarios, is the closed gap score. This metric assigns to a meta-solver a value in $(-\infty, 1]$ proportional to how much it closes the gap between the best individual solver available, or single best solver (SBS), and the virtual best solver (VBS), i.e., an oracle-like meta-solver always selecting the best individual solver. The closed gap is actually a “meta-metric“, defined in terms of another evaluation metric m to minimize, which in this case is the scoring metric defined earlier. Formally, if (I, S, τ) is a scenario then

$$m(i, \text{VBS}, \tau) = \min\{m(i, s, \tau) \mid s \in S\} \quad \text{for each } i \in I,$$

and

$$\text{SBS} = \arg \min_{s \in S} \sum_{i \in I} m(i, s, \tau).$$

With these definitions, *Closed-gap* can be defined as follows: Let (\mathcal{I}, S, τ) be a scenario and

$$m : \mathcal{I} \times (S \cup \{S, \text{VBS}\}) \times [0, \tau] \rightarrow \mathbb{R}$$

an evaluation metric to minimize for that scenario, where S is a meta-solver over the solvers of S . Let

$$m_\sigma = \sum_{i \in \mathcal{I}} m(i, \sigma, \tau) \quad \text{for } \sigma \in \{S, \text{SBS}, \text{VBS}\}.$$

The closed gap of S with respect to m on that scenario is

$$\frac{m_{\text{SBS}} - m_S}{m_{\text{SBS}} - m_{\text{VBS}}}.$$

The assumption $m_{\text{VBS}} > m_{\text{SBS}}$ is required, i.e., no single-solver can be the VBS (otherwise, no algorithm selection would be needed, given that its objective is to reach the VBS). Unlike other scores, the closed gap is designed specifically for meta-solvers. Applying it to individual solvers would assign 0 to the SBS and a negative score to the remaining solvers, proportional to their performance difference with respect to the SBS and the gap $m_{\text{SBS}} - m_{\text{VBS}}$, which makes little sense for individual solvers, as it wouldn't reflect their actual performance overall.

2.1.6 Experiment Setup

We have defined both the structure of the queries posed to the LLMs (Section 2.1.4) and the way in which these queries are formulated (Section 2.1.3). The remaining challenge is to evaluate them automatically over the full set of selected instances. To this end, we designed an automated testing pipeline that parallelizes execution by assigning one thread per LLM. For each model, requests are issued sequentially, with five requests per problem, each containing a MiniZinc model and a single instance encoded as shown in Figure 2.1.

Despite preliminary prompt engineering and model filtering, several MiniZinc models, particularly their associated data files, still exceed the providers' rate limits. Since these limits are strict, additional mechanisms were required to prevent limit violations while still allowing evaluation over the complete instance set.

Script Manipulation

The most direct way to address oversized requests is to reduce their length. As the prompt itself was already minimal, this required direct manipulation of the MiniZinc model (`.mzn`) and data files.

A first step consisted in removing all non-essential elements, such as comments (starting with `% [3]`), tabs, and unnecessary whitespace. While this helps reduce token usage and standardizes script formatting, it is insufficient on its own. The main contributor to token overflow is the presence of large data arrays, which not only increase message length but may also pollute the context, “distracting” the LLM from the most relevant information [22].

To mitigate this issue, data arrays were truncated to a fixed maximum length of 30 elements, with an inline comment indicating the original size:

```
[e1, e2, ..., e30// array too long to display, dimensions: (150)]
```

While effective for simple arrays of scalar values, this approach does not account for the complexity of individual elements and performs poorly on more structured data. For this reason, a second truncation mechanism was introduced based on raw character length. Arrays exceeding 90 characters were truncated accordingly, using the same annotation to preserve information about the original size.

Custom Delays

Since each experiment involves multiple problems and multiple sequential requests per LLM, rate limits can still be exceeded even when individual requests are within bounds. To handle this, custom delays were introduced into the experiment orchestration logic.

When an error message is received, for example:

```
Error code: 413 - Request too large for model 'openai/gpt-oss-120b' in organization 'org_01k9qqesvte4d9h5jnhmzvbm4' service tier 'on_demand' on tokens per minute (TPM): Limit 8000, Requested 8939, please reduce your message size and try again.
Need more tokens? Upgrade to Dev Tier today at https://console.groq.com/settings/billing
```

```
Error code: 429 - Rate limit reached for model 'openai/gpt-oss-120b' in organization 'org_01k9qqesvte4d9h5jnhmzvbm4' service tier 'on_demand' on tokens per day (TPD): Limit 200000, Used 193047, Requested 10632. Please try again in 26m29.328s. Need more tokens? Upgrade to Dev Tier today at https://console.groq.com/settings/billing
```

Its code is inspected. Errors 413 and 429 indicate that a rate limit has been exceeded. The error message is then parsed to identify the specific limit involved. If the limit concerns tokens per minute (TPM) or requests per minute (RPM), the system pauses execution for 60 seconds before retrying. If the exceeded limit is tokens per day (TPD) or requests per day (RPD), the message is further analyzed to extract the cooldown duration, typically expressed in the form $XXh, XXm, XX.XXs$ where h stands for hours, m for minutes and s for seconds. The required delay is then computed from this value, after which the request is retried.

2.2 Experiments

In this section we are gonna show and explain al the experiments that led to the final choice of the context information and overall setup of the agentic solver. ...structure description ...

2.2.1 Preliminary tests

The first experiments were conducted using the unedited problems. Each LLM was provided with the original MiniZinc model (.mzn) together with the corresponding instance data (in either .dzn or .json format), following the prompt structure defined in Section 2.1.3.

Model	Single Score	Parallel Score	Closed Gap
gemini-2.5-flash-lite	64.363	69.040	-1.047
gemini-2.5-flash	60.962	69.426	-1.330
moonshotai/kimi-k2-instruct-0905	59.680	66.186	-1.436
moonshotai/kimi-k2-instruct	58.609	65.816	-1.525
openai/gpt-oss-120b	58.166	64.508	-1.562
openai/gpt-oss-20b	57.154	63.329	-1.646
meta-llama/llama-4-maverick-17b-128e-instruct	56.297	63.549	-1.717
meta-llama/llama-4-scout-17b-16e-instruct	54.305	57.815	-1.883
gemini-2.0-flash	43.413	53.748	-2.788
qwen/qwen3-32b	42.117	48.018	-2.895
gemini-2.5-pro	37.641	41.594	-3.267
llama-3.1-8b-instant	36.082	56.228	-3.397
groq/compound-mini	25.423	29.623	-4.282
llama-3.3-70b-versatile	7.455	9.496	-5.775
groq/compound	5.197	8.246	-5.963

Table 2.3: Initial tests giving plain scripts to all the LLMs, In this table:column "Model" contains the names of each tested LLM, "Total Score" represents the sum of the score reached in every instance, in this table "Total Score" represents the sum of the score reached in every instance, score was calculated by summing the performance of what was suggested to be the best solver on the given instance by each of the LLMs, "Parallel Score" represents the sum of the score reached in every instance, score was calculated in parallel-solver setup, so taking the 3 best solvers given by the LLM, calculate the score of all the 3, and take the maximum out of the three, and finally "Closed Gap" displays the closed gap score calculated over "Single Score" by using the formula explained in Section 2.1.5.

As reported in Table 2.3, (and more in depth in Table 2.11, Table 2.10 and Table 2.12) for both single-solver evaluation and parallel-solver evaluation, the scores are all under 70, meaning a lower performance than 3 of the single solvers from free category, and 4 of the single

solvers from open category, and clearly a negative closed gap for all of the LLMs. While part of this outcome can be attributed to the deliberately simple formulation of the requests, a major limiting factor is the presence of strict rate limits, which prevent many instances from being processed by some, if not all, of the LLMs, due to the script length alone exceeding TPM (showed in Table 2.2 and Table 2.1). This problem is predominant in problems with large instance data, such as: `ihtc-2024-marte` or `gt-sort`

These constraints motivated both the adoption of script manipulation techniques, as described in Section 2.1.6, and, simple time issues due to tests taking even up to 48 hours, the decision to restrict subsequent experiments to the five best-performing LLMs identified in this preliminary phase, namely: `gpt-oss-120b`, `gemini-2,5-flash`, `gemini-2,5-flash-lite`, `kimi-k2-instruct-0905` and `kimi-k2-instruct`.

2.2.2 Single Request Experiments

After establishing a stable experimental pipeline, we repeated the evaluation on the full set of 100 instances. All experiments in this phase adopt a single-request setup, where each prompt is processed independently: the LLM receives the input and produces an answer without access to any prior interaction history or retained context.

Base Setup

As a baseline, the LLMs were first evaluated under the same conditions as the preliminary experiments, using only the raw MiniZinc and data scripts.

Model	Single Score	Parallel Score	Closed Gap
openai/gpt-oss-120b	74.488	82.227	-0.206
moonshotai/kimi-k2-instruct-0905	71.623	82.657	-0.444
moonshotai/kimi-k2-instruct	70.939	83.268	-0.501
gemini-2.5-flash-lite	70.145	80.741	-0.567
gemini-2.5-flash	69.763	79.105	-0.598

Table 2.4: Tests on sanitized scripts given to the 5 best performing LLMs, columns content is calculated as in Table 2.3.

As shown in Table 2.4 and Table 2.13, performance in the parallel-solver evaluation is generally strong. All tested LLMs outperform every individual solver except `or-tools_cp-sat-par`, which corresponds to the single best solver (SBS) in the open category and remains clearly dominant, with a substantial margin over both the LLM-based meta-solvers and the remaining individual solvers.

Greater variability emerges in the single-solver evaluation (Table 2.14). In this case, only `gpt-oss-120b` consistently outperforms all individual solvers other than the SBS in the free category (`or-tools-cp-sat-free`), as will be later reported in Table 2.26 and Figure 2.5b. The remaining LLMs still achieve competitive results compared to most standalone solvers, but a significant performance gap remains, as highlighted by the closed-gap scores, also reported more in depth in Table 2.15.

Problem Description

The results of the baseline experiments indicate that further improvements to the Agentic Solver are necessary. A natural approach is to provide the LLM with additional contextual information. However, as discussed previously, excessively large contexts can be counterproductive, potentially distracting the model and degrading performance rather than improving it [15, 22]. This trade-off motivates a more careful investigation into which types of information are most beneficial for LLM-based solver selection.

As a first step, we augmented the prompt with a concise problem description (PD): a short textual summary of the MiniZinc model’s semantics. These descriptions were automatically generated using another LLM (GPT-5.1) and subsequently refined manually to correct minor inaccuracies, e.g.:

- **atsp**: “Scheduling and resource allocation problem involving moulds, colors, and production jobs. The goal is to minimize makespan, tardiness, and waste while respecting compatibility and demand constraints.”
- **black-hole**: “A constraint model for solving the Black Hole Patience solitaire game. Cards must be arranged so that the sequence follows game rules using global constraints.”

The PD was incorporated into the prompt structure (Figure 2.1) as:

Model	Single Score	Parallel Score	Closed Gap
moonshotai/kimi-k2-instruct-0905	72.605	83.182	-0.362
openai/gpt-oss-120b	70.741	83.250	-0.517
gemini-2.5-flash-lite	69.043	82.621	-0.658
moonshotai/kimi-k2-instruct	67.555	81.890	-0.782
gemini-2.5-flash	49.897	59.154	-2.249

Table 2.5: Test on sanitized scripts combined with textual problem description. Columns are calculated as in Table 2.3.

Prompt Structure

Prompt description:

...Textual problem description ...

MiniZinc model:

...Minimizing problem model (.mzn content) ...

MiniZinc data:

...Instance relative data (.dzn or .json content) ...

The goal is to determine which constraint programming solver would be best suited for this problem, considering the following options:

- s_1 ,
- s_2 ,
- ...
- s_n where $s_1 \dots s_n \in \text{SolverList}$.

Answer only with the name of the 3 best solvers inside square brackets separated by comma and nothing else.

As shown in Table 2.5, single-solver performance consistently degrades, while parallel-solver evaluation shows modest improvements, with three out of the five tested LLMs achieving better results than in the base configuration. This divergence suggests that additional context can aid diversification in solver selection, even if, in this case, it does not reliably improve the choice of an optimal solver.

As can be deduced from the results, all the closed gap scores are negatives even after improving the context.

2.2.3 Multi-turn Experiments

What we discussed so far indicate that the contextual information provided to the LLMs is either insufficient or, in some cases, not beneficial overall. A natural next step is therefore to explore alternative forms of context. However, this introduces two practical issues. First, the single-request setup already operates close to the maximum allowed tokens per minute (TPM), making it infeasible to simply add more information without violating rate limits. Second, the existing setup is inefficient in terms of token usage, as it repeatedly supplies redundant information.

More specifically, for each problem we evaluate five different instances, each with distinct data, while the underlying MiniZinc model and the associated problem description remain unchanged. Re-sending this invariant information with every request unnecessarily consumes tokens. Given the limited API resources available, addressing both constraints is essential. To this end, we transitioned to a multi-turn (chat-like) experimental setup.

Setup Explanation

The core idea of the multi-turn setup is to partition the interaction using role-based formatting [23]. In the context of LLMs, messages are explicitly associated with roles, typically **system**, **user**, and **assistant**. Which helps the LLM distinguish between instructions, inputs, and generated outputs, while also maintaining conversational state across turns.

In our experiments, the **system** role is used to convey all invariant and high-level information, namely the MiniZinc model (**.mzn**) content, the textual descriptions, and the expected format of the answers. The **user** role is then reserved for instance-specific inputs, containing the data associated with each instance (in **.dzn** or **.json** format). This separation allows us to avoid repeatedly transmitting redundant context, significantly reducing token consumption per instance.

An additional advantage of the multi-turn setup is that it enables the handling of larger instance data by distributing content across multiple messages, while relying on the LLM’s ability to retain previously supplied information within the same conversation. As a result, the system can accommodate longer and more complex inputs without exceeding rate limits.

Solvers Description

The increased efficiency of the multi-turn setup also makes it possible to enrich the contextual information with new textual data: solver descriptions. For each solver under consideration, a short textual description was generated and provided to the LLM within the **system** prompt, with the aim of improving the model’s awareness of the available solver options and their respective characteristics.

To better understand the importance of this information, we experimented with two different configurations: one in which solver descriptions were combined with all previously provided contextual elements, and another in which the solvers descriptions constituted the only additional textual information alongside the MiniZinc model and the instance data. This design allows us to assess the contribution of solver-specific knowledge to the overall performance of the Agentic Solver.

Model	Single Score	Parallel Score	Closed Gap
moonshotai/kimi-k2-instruct	76.964	82.236	0.000
moonshotai/kimi-k2-instruct-0905	76.964	82.489	0.000
gemini-2.5-flash	71.687	83.137	-0.439
openai/gpt-oss-120b	70.974	78.800	-0.498
gemini-2.5-flash-lite	54.714	77.747	-1.849

Table 2.6: Test with sanitized scripts combined with solvers description in a multi turn setup. Columns are calculated as in Table 2.3.

In the parallel-solver evaluation (Table 2.19), providing only solver descriptions does not lead to systematic improvements. The sole exception is **gemini-2.5-flash** although its score is still under that of the single best solver (SBS) in the open category.

On the other hand, the effects are more pronounced in the single-solver evaluation, displayed in Table 2.6Table 2.20. Two models, **moonshotai/kimi-k2-instruct-0905** and **moonshotai/kimi-k2-instruct**, exhibit a substantial improvement, reaching the SBS score and thus achieving a closed-gap value of zero for the first time. A closer inspection of their outputs, however, reveals that this result is achieved by consistently selecting the same solver, namely **or-tools_cp-sat-free**, which is itself the SBS. This behavior effectively bypasses the decision-making role of the LLM, thereby undermining the intended purpose of employing an LLM in the first place.

Solver Description and Problem Description

Following this observation, we evaluated the configuration combining both forms of textual context: solver descriptions and problem descriptions. In this setup, each LLM is provided with the largest amount of contextual information until now.

Model	Single Score	Parallel Score	Closed Gap
openai/gpt-oss-120b	77.261	80.296	0.025
moonshotai/kimi-k2-instruct-0905	76.964	82.219	0.000
moonshotai/kimi-k2-instruct	74.464	80.417	-0.208
gemini-2.5-flash	73.552	83.651	-0.284
gemini-2.5-flash-lite	63.063	78.148	-1.155

Table 2.7: Test with sanitized scripts combined with both solvers description, and problem description in a multi turn setup. Columns are calculated as in Table 2.3.

In the parallel-solver evaluation, this configuration yields the highest scores observed so far, again driven primarily by **gemini-2.5-flash**. However, otehr LLMs score is slightly lower than in the previous setups, and all of the reached scores are still under the open-category SBS.

In the single-solver evaluation, **moonshotai/kimi-k2-instruct-0905** continues to default to selecting the SBS exclusively. Nevertheless, improvements emerge for two other models, namely **gemini-2.5-flash** and **gpt-oss-120b**. In particular, **gpt-oss-120b** achieves the first strictly positive closed-gap score, surpassing **or-tools_cp-sat-free**.

Basic Tests Evaluation

In this initial testing phase, a positive closed gap has been achieved, showing an agentic solver with better performance than the single best solver. Moreover, when we put the perfor-

mance respect to the “non-best solvers“, these primitive configurations of agentic solvers still hold fairly competitive results.

To better display single LLMs performances, all variants scores are put together against one another, using histograms for better visualization in Figure 2.3. In Figure 2.3a for parallel-solver evaluation, Figure 2.3b for single-solver evaluation and Figure 2.3c for closed gap evaluation.

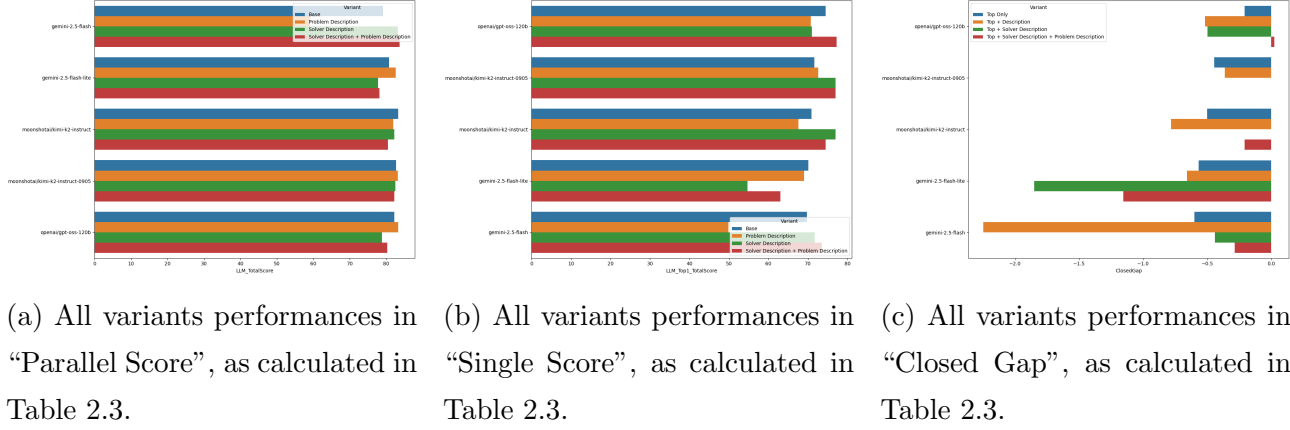


Figure 2.3: Histograms displaying the performances of all the solver variants in “Single Score” evaluation, “Parallel Score” evaluation and “Closed Gap” evaluation.

To give results another point of view, the performance of each agentic solver configuration is displayed against the single solvers of the corresponding category. In Table 2.25 and Figure 2.5a are shown the performance of all the configurations, scored with parallel-solvers evaluation, when put against single solvers from the open category. On the other hand, looking at Table 2.26 and Figure 2.5b, the resulting score of single-solver evaluation of all the variants is put against all the single solvers from free category.

2.2.4 Feature Employment

In the last section, we exposed the importance of textual informations, and giving a richer context to the LLM instead of relying on its understanding of a `.mzn` script. Even though a closed gap was already reached in previous experiments, relying on context information such as the problem description is a strong limitation for the solver, given those data need to be supervised, if not entirely rewritten by hand, due to the high variability, especially when extracted on new problems. This limitation highlights the necessity of a mean to extract informations automatically, in a controlled and predictable way.

Another problem is the one concerning script dimensions: programs with longer scripts need some techniques like sanitization, as previously stated in Section 2.1.6, these type of technique, makes it possible to work with longer scripts and context data. But is still penalizing towards longer problems, raising the risk of hallucination [24], due to the forced removal of elements in data, leaving incomplete arrays as input.

For these tasks we decided to employ a feature-extractor [25], which allows to extract an extensive set of 95 features from a Constraint (Satisfaction/Optimization) Problem defined in possibly different modelling languages: MiniZinc, FlatZinc or XCSP. Designed to be independent from the particular machine on which it is run as well as from the specific global redefinitions of a given solver. The employed version was already used in other projects [26] [27] [28].

Tool Description

The tool `mzn2feat` is designed to extract a set of 155 features from a MiniZinc model. Of these, 144 are static features derived through syntactic analysis of the source problem instance, while 11 are dynamic features obtained via a short execution of the Gecode solver. Due to the complexity of the MiniZinc language, particularly the presence of control-flow constructs, direct extraction of syntactic features from MiniZinc models is nontrivial. To address this, models are first compiled into FlatZinc, a lower-level language whose syntax is largely a subset of MiniZinc. This translation is performed using the `mzn2fzn` tool provided within the MiniZinc toolchain.

The compilation step employs Gecode-specific redefinitions of global constraints. This preserves information about the presence and type of global constraints without decomposing them into primitive constraints. Such preservation is relevant because, in the absence of solver-specific redefinitions, certain global constraints (e.g., `alldifferent`) when decomposed into sets of simpler constraints, from which the original high-level constraint cannot be uniquely reconstructed.

Static feature extraction from the resulting FlatZinc model is performed using `fzn2feat`, a parser implemented with Flex and Bison. Dynamic features are obtained by executing the Gecode FlatZinc interpreter (`fz`) for a fixed time budget of two seconds on the compiled model.

In summary, given a MiniZinc model M , the `mzn2feat` workflow consists of three stages. First, M is translated into a FlatZinc model F_M using Gecode global constraint redefinitions. Second, static features are extracted from F_M via `fzn2feat`. Third, dynamic features are extracted from F_M through a bounded run of the Gecode interpreter. The static feature extraction stage is applicable to any FlatZinc model, although solver-specific redefinitions that are not recognized may be ignored. The static and dynamic feature extraction procedures are independent, allowing them to be executed in parallel or in arbitrary order. For example, static feature computation may be omitted if the instance is solved during the dynamic feature collection phase.

For a better understanding of all of the features, refer to Table 2.27 for the description by category, and to 2.1 as an example output.

Testing and Results

To understand the potential of the feature extractor we tried to use the pretty-print output (like the one in 2.1) combined with the context informations given until now. So we tried

testing:

- Only `mzn2feat` output.
- `mzn2feat` output and problem text description.
- `mzn2feat` output, problem text description and solvers text description.

And then tried those setup, adding first the `.mzn` problems only, and finally also the sanitized data scripts (`.dzn/.json`).

Variant	Single Score	Parallel Score	Closed Gap
Features + Solver Description + Problem Description	75.659731	83.731582	-0.108399
Features + Solver Description	75.390197	82.994278	-0.130793
Features + Problem Description	74.829813	83.289246	-0.177354
Features	73.600955	82.259760	-0.279455
Features + <code>.mzn</code> + Problem Description	71.916389	78.530605	-0.419420
Features + <code>.mzn</code>	70.368365	78.371276	-0.548041
Features + <code>.mzn</code> + Instance Data	68.422708	74.803213	-0.709699
Features + <code>.mzn</code> + Instance Data + Solver Description	63.585459	66.346631	-1.111610
Features + <code>.mzn</code> + Instance Data + Problem Description	62.516550	69.284744	-1.200422
Features + <code>.mzn</code> + Solver Description	61.687990	66.874762	-1.269264
Features + <code>.mzn</code> + Solver Description + Problem Description	58.576543	64.774159	-1.527784
Features + <code>.mzn</code> + Instance Data + Solver Description + Problem Description	54.188182	59.410873	-1.892398

Table 2.8: Test with all the possible combinations involving features extracted using `mzn2feat` [25], all the tests have been made using `gpt-oss-120b` as it’s the only model that produced a positive closed gap until this point. Columns are calculated as in Table 2.3.

As displayed in Table 2.8, the best performing instances are the ones combining features and textual informations, while adding `.mzn` or instance data is consistently lowering score, most probably because of some problems explained earlier such as context rot [15].

It should also be noted that The best performing variants are all right behind the best performing one from previous tests but still, the best 3 reached an higher score than the second best “simple” variant.

2.2.5 Sampling Temperature Tuning

The testing of this initial phase involved the use of sampling temperature tuning. Sampling temperature is a hyperparameter of an LLM used in a temperature-based sampling process. It controls the randomness of the model’s output at inference time [29].

During each step of an LLM’s decoding process, the LLM uses the previous tokens to choose the next output token. The final layer of the LLM uses a softmax function to convert raw scores (logits) into probabilities.

In greedy sampling, the model will always choose the most likely next token. However, for probabilistic sampling, the next token is selected from a probability distribution.

Temperature sampling is a modification to the softmax function, which adjusts the resulting probability mass functions. In this modified softmax function, v_k is the k -th vocabulary token, l_k is the token’s logit, and τ is a constant temperature:

$$\Pr(v_k) = \frac{e^{l_k/\tau}}{\sum_i e^{l_i/\tau}}$$

A lower temperature makes the output of the LLM more deterministic, thus favoring the most likely predictions. This conservativeness is captured by the model’s tendency to produce more repetitive, focused, and less diverse output based on the patterns most commonly seen in the training data [30] [31].

A higher temperature increases the randomness of the output, thus favoring more “creative” predictions. This creativity is captured by the model’s willingness to explore more unconventional and less likely outputs. Higher temperatures can lead to novel text, diverse ideas, and creative solutions to problems [30] [31].

In the context of problem-solving, temperature can be seen as a trade-off between exploring possible solutions within the solution space and exploiting probable solutions; lower temperatures tend to exploit probable solutions, whereas higher temperatures explore the solution space more broadly.

Choosing Sampling Temperatures

Low sampling temperatures are recommended for tasks requiring precision and factual accuracy, such as technical writing, code generation, or question-answering [32]. However, higher temperatures are recommended for tasks requiring creativity, such as writing poetry, creating stories, or brainstorming.

Higher temperatures also increase the probability of model hallucination where an LLM produces statistically probable responses that are factually incorrect or nonsensical. As a result, optimal temperature selection is also a balance between creativity and hallucination [24].

As for tests with features employment, the only large language model that will be employed for testing is **gpt-oss-120b**. Being the LLM employed via Groq API [12], we decided to opt for the temperature pre-sets displayed in Table 2.9, also indicated in the documentations [33], as a testing ground.

Testing and Results

As anticipated, the tests were all made only using **gpt-oss-120b** for LLM. Temperature sampling tests were made on the 5 best performing variants, namely:

1. **.mzn** scripts, instance data and text description for both solvers and problems.
2. Features extracted through **mzn2feat** and text description for both solvers and problems.
3. Features extracted through **mzn2feat** and text description for solvers.

Scenario	Temp	Comments
Factual Q&A	0.2	Keeps dates & numbers stable
Data extraction (JSON)	0.0	Deterministic keys/values
Creative copywriting	0.8	Vivid language, fresh ideas
Brainstorming list	0.7	Variety without nonsense
Long-form code	0.3	Fewer hallucinated APIs

Table 2.9: Temperature setting suggestions from Groq documentation [33]. In the leftmost column is presented the basic scenario in which the following settings are more indicated, followed by the sampling temperature value, and finally a short comment on the expected behaviour from the LLM with the given setting.

4. Features extracted through `mzn2feat` and text description for problems.
5. `.mzn` scripts and instance data

The limitation to those variants only, was simply due to resources limitations from the rate limits on the model.

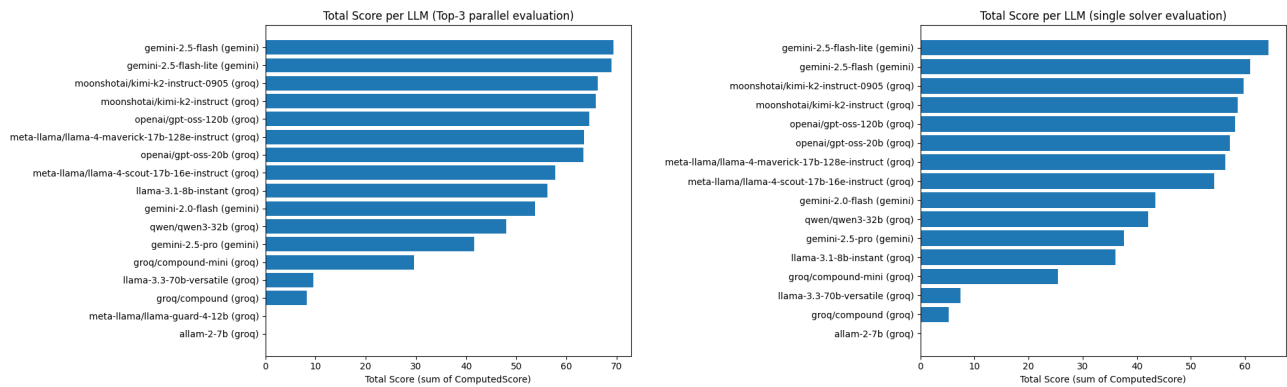
Appendix

Model	Total Score	Instances Covered	Average Score
gemini-2.0-flash	53.748	67	0.802
gemini-2.5-flash	69.426	86	0.807
gemini-2.5-flash-lite	69.040	85	0.812
gemini-2.5-pro	41.594	51	0.815
allam-2-7b	0.0	10	0.0
groq/compound	8.246	9	0.916
groq/compound-mini	29.623	35	0.846
llama-3.1-8b-instant	56.228	72	0.780
llama-3.3-70b-versatile	9.496	10	0.949
meta-llama/llama-4-maverick-17b-128e-instruct	63.548	72	0.882
meta-llama/llama-4-scout-17b-16e-instruct	57.814	69	0.837
meta-llama/llama-guard-4-12b	0.0	77	0.0
moonshotai/kimi-k2-instruct	65.816	75	0.877
moonshotai/kimi-k2-instruct-0905	66.186	75	0.882
openai/gpt-oss-120b	64.508	74	0.871
openai/gpt-oss-20b	63.328	75	0.844
qwen/qwen3-32b	48.018	65	0.738

Table 2.10: Initial tests giving plain scripts to all the LLMs, In this table: column "Model" contains the names of each tested LLM, "Total Score" is the equivalent of "Parallel Score" as explained in Table 2.3, "Instances Covered" contains the amount of instances that gave a viable result so the ones that did not exceed the token limitations portrayed in Table 2.1 and Table 2.2, "Average Score" represents $\frac{TotalScore}{InstancesCovered}$ to show the solver performance in the evaluated instances.

Model	Total Score	Instances Covered	Average Score
gemini-2.5-flash-lite	64.363	85	0.794
gemini-2.5-flash	60.962	85	0.734
moonshotai/kimi-k2-instruct-0905	59.680	75	0.817
moonshotai/kimi-k2-instruct	58.609	75	0.837
openai/gpt-oss-120b	58.166	74	0.796
openai/gpt-oss-20b	57.154	75	0.828
meta-llama/llama-4-maverick-17b-128e-instruct	56.297	72	0.804
meta-llama/llama-4-scout-17b-16e-instruct	54.305	69	0.798
gemini-2.0-flash	43.412	67	0.700
qwen/qwen3-32b	42.116	65	0.779
gemini-2.5-pro	37.640	51	0.738
llama-3.1-8b-instant	36.082	72	0.546
groq/compound-mini	25.422	35	0.726
llama-3.3-70b-versatile	7.454	10	0.745
groq/compound	5.197	9	0.577
allam-2-7b	0.0	4	0.0

Table 2.11: Initial tests giving plain scripts to all the LLMs, in this table “Total Score” is the equivalent of “Single Score” as explained in Table 2.3. “Instances Covered” and “Average Score” are calculated the same way as in Table 2.10



(a) Scores obtained by all of the considered LLM from parallel-solver evaluation, “Total Score” is the equivalent of “Single Score” as explained in Table 2.3.

(b) Scores obtained by all of the considered LLM from single-solver evaluation, “Total Score” is the equivalent of “Parallel Score” as explained in Table 2.3

Figure 2.4: Histograms displaying performances of All the LLMs with plain scripts

model	InstancesCovered	AS	SBS	VBS	ClosedGap
gemini-2.5-flash-lite	85	64.363	76.964	89.000	-1.047
gemini-2.5-flash	85	60.962	76.964	89.000	-1.330
moonshotai/kimi-k2-instruct-0905	75	59.680	76.964	89.000	-1.436
moonshotai/kimi-k2-instruct	75	58.609	76.964	89.000	-1.525
openai/gpt-oss-120b	74	58.166	76.964	89.000	-1.562
openai/gpt-oss-20b	75	57.154	76.964	89.000	-1.646
meta-llama/llama-4-maverick-17b-128e-instruct	72	56.297	76.964	89.000	-1.717
meta-llama/llama-4-scout-17b-16e-instruct	69	54.305	76.964	89.000	-1.883
gemini-2.0-flash	67	43.413	76.964	89.000	-2.788
qwen/qwen3-32b	65	42.117	76.964	89.000	-2.895
gemini-2.5-pro	51	37.641	76.964	89.000	-3.267
llama-3.1-8b-instant	72	36.082	76.964	89.000	-3.397
groq/compound-mini	35	25.423	76.964	89.000	-4.282
llama-3.3-70b-versatile	10	7.455	76.964	89.000	-5.775
groq/compound	9	5.197	76.964	89.000	-5.963

Table 2.12: Initial tests giving plain scripts to all the LLMs, in this table: “Instances Covered” is the same as in Table 2.10, “AS” is the same as the “Single Score” explained in Table 2.3, “SBS” displays the sum of all the scores obtained by the single best solver (namely, `or-tools_cp-sat-free`) in every instance, and “VBS” displays the score obtained with the use of an hypothetical virtual best solver, giving the maximum obtainable score on every instance. Finally, “Closed Gap” is calculated as in Table 2.3

Model	Total Score	Instances Covered	Average Score
gemini-2.5-flash	79.105	100	0.791
gemini-2.5-flash-lite	80.740	100	0.807
moonshotai/kimi-k2-instruct	83.268	100	0.832
moonshotai/kimi-k2-instruct-0905	82.656	100	0.826
openai/gpt-oss-120b	82.226	100	0.822

Table 2.13: Tests on sanitized scripts given to the 5 best performing LLMs, parallel-solver evaluation. Columns are calculated as in Table 2.10

Model	Total Score	Instances Covered	Average Score
openai/gpt-oss-120b	74.488	100	0.744
moonshotai/kimi-k2-instruct-0905	71.622	100	0.753
moonshotai/kimi-k2-instruct	70.939	100	0.723
gemini-2.5-flash-lite	70.145	100	0.738
gemini-2.5-flash	69.763	98	0.742

Table 2.14: Tests on sanitized scripts given to the 5 best performing LLMs, single-solver evaluation. Columns are calculated as in Table 2.11

Model	Instances Covered	AS	SBS	VBS	Closed Gap
openai/gpt-oss-120b	100	74.488	76.964	89.0	-0.205
moonshotai/kimi-k2-instruct-0905	100	71.622	76.964	89.0	-0.443
moonshotai/kimi-k2-instruct	100	70.939	76.964	89.0	-0.500
gemini-2.5-flash-lite	100	70.145	76.964	89.0	-0.566
gemini-2.5-flash	98	69.763	76.964	89.0	-0.598

Table 2.15: Tests on sanitized scripts given to the 5 best performing LLMs, closed gap. Columns are calculated as in Table 2.12

Model	Total Score	Instances Covered	Average Score
gemini-2.5-flash	59.154	75	0.788
gemini-2.5-flash-lite	82.620	100	0.826
moonshotai/kimi-k2-instruct	81.889	100	0.818
moonshotai/kimi-k2-instruct-0905	83.182	100	0.831
openai/gpt-oss-120b	83.249	100	0.832

Table 2.16: Test on sanitized scripts combined with textual problem description, parallel-solver evaluation. Columns are calculated as in Table 2.10

Model	Total Score	Instances Covered	Average Score
moonshotai/kimi-k2-instruct-0905	72.605	100	0.748
openai/gpt-oss-120b	70.740	100	0.721
gemini-2.5-flash-lite	69.042	100	0.719
moonshotai/kimi-k2-instruct	67.554	100	0.718
gemini-2.5-flash	49.896	73	0.723

Table 2.17: Test on sanitized scripts combined with textual problem description, single-solver evaluation. Columns are calculated as in Table 2.11

Model	Instances Covered	AS	SBS	VBS	Closed Gap
moonshotai/kimi-k2-instruct-0905	100	72.605	76.964	89.0	-0.362
openai/gpt-oss-120b	100	70.740	76.964	89.0	-0.517
gemini-2.5-flash-lite	100	69.042	76.964	89.0	-0.658
moonshotai/kimi-k2-instruct	100	67.554	76.964	89.0	-0.781
gemini-2.5-flash	73	49.896	76.964	89.0	-2.248

Table 2.18: Test on sanitized scripts combined with textual problem description, closed gap. Columns are calculated as in Table 2.12

Model	Total Score	Instances Covered	Average Score
gemini-2.5-flash	83.137	100	0.831
gemini-2.5-flash-lite	77.746	100	0.777
moonshotai/kimi-k2-instruct	82.236	100	0.822
moonshotai/kimi-k2-instruct-0905	82.488	100	0.824
openai/gpt-oss-120b	78.799	100	0.787

Table 2.19: Test with sanitized scripts combined with solvers description in a multi turn setup, parallel-solver evaluation. Columns are calculated as in Table 2.10

Model	Total Score	Instances Covered	Average Score	
moonshotai/kimi-k2-instruct	76.964	100	0.769	
moonshotai/kimi-k2-instruct-0905	76.964	100	0.769	
gemni	gemini-2.5-flash	71.686	100	0.716
openai/gpt-oss-120b	70.974	100	0.716	
gemni	gemini-2.5-flash-lite	54.713	100	0.552

Table 2.20: Test with sanitized scripts combined with solvers description in a multi turn setup, single-solver evaluation. Columns are calculated as in Table 2.11

Model	Instances Covered	AS	SBS	VBS	Closed Gap
moonshotai/kimi-k2-instruct	100	76.964	76.964	89.0	0.0
moonshotai/kimi-k2-instruct-0905	100	76.964	76.964	89.0	0.0
gemini-2.5-flash	100	71.686	76.964	89.0	-0.438
openai/gpt-oss-120b	100	70.974	76.964	89.0	-0.497
gemini-2.5-flash-lite	100	54.713	76.964	89.0	-1.848

Table 2.21: Test with sanitized scripts combined with solvers description in a multi turn setup, closed gap. Columns are calculated as in Table 2.12

Model	Total Score	Instances Covered	Average Score
gemini-2.5-flash	83.650	100	0.836
gemini-2.5-flash-lite	78.147	100	0.781
moonshotai/kimi-k2-instruct	80.417	99	0.812
moonshotai/kimi-k2-instruct-0905	82.218	100	0.822
openai/gpt-oss-120b	80.295	100	0.802

Table 2.22: Test with sanitized scripts combined with both solvers description, and problem description in a multi turn setup, parallel-solver evaluation. Columns are calculated as in Table 2.10

Model	Total Score	Instances Covered	Average Score
openai/gpt-oss-120b	77.260	100	0.780
moonshotai/kimi-k2-instruct-0905	76.964	100	0.769
moonshotai/kimi-k2-instruct	74.464	99	0.752
gemini-2.5-flash	73.551	100	0.750
gemini-2.5-flash-lite	63.062	100	0.630

Table 2.23: Test with sanitized scripts combined with both solvers description, and problem description in a multi turn setup, single-solver evaluation. Columns are calculated as in Table 2.11

Model	Instances Covered	AS	SBS	VBS	Closed Gap
openai/gpt-oss-120b	100	77.260	76.964	89.0	0.024
moonshotai/kimi-k2-instruct-0905	100	76.964	76.964	89.0	0.0
moonshotai/kimi-k2-instruct	99	74.464	76.964	89.0	-0.207
gemini-2.5-flash	100	73.551	76.964	89.0	-0.283
gemini-2.5-flash-lite	100	63.062	76.964	89.0	-1.155

Table 2.24: Test with sanitized scripts combined with both solvers description, and problem description in a multi turn setup, closed gap. Columns are calculated as in Table 2.12

Type	Solver	Total Score
Solver	or-tools_cp-sat-par	88.117
LLM Variant	gemini-2.5-flash (Solvers Description + Problem Description)	83.650
LLM Variant	moonshotai/kimi-k2-instruct	83.268
LLM Variant	openai/gpt-oss-120b (Problem Description)	83.249
LLM Variant	moonshotai/kimi-k2-instruct-0905 (Problem Description)	83.182
LLM Variant	gemini-2.5-flash (Solvers Description)	83.137
LLM Variant	moonshotai/kimi-k2-instruct-0905	82.656
LLM Variant	gemini-2.5-flash-lite (Problem Description)	82.620
LLM Variant	moonshotai/kimi-k2-instruct-0905 (Solvers Description)	82.488
LLM Variant	moonshotai/kimi-k2-instruct (Solvers Description)	82.236
LLM Variant	openai/gpt-oss-120b	82.226
LLM Variant	moonshotai/kimi-k2-instruct-0905 (Solvers Description + Problem Description)	82.218
LLM Variant	moonshotai/kimi-k2-instruct (Problem Description)	81.889
LLM Variant	gemini-2.5-flash-lite	80.740
LLM Variant	openai/gpt-oss-120b (Solvers Description + Problem Description)	80.295
LLM Variant	gemini-2.5-flash	79.105
LLM Variant	openai/gpt-oss-120b (Solvers Description)	78.799
LLM Variant	gemini-2.5-flash-lite (Solvers Description + Problem Description)	78.147
LLM Variant	gemini-2.5-flash-lite (Solvers Description)	77.746
Solver	chuffed-free	74.819
Solver	picatsat-free	70.647
Solver	huub-free	68.784
Solver	gurobi-par	61.081
Solver	cplex-par	60.301
Solver	choco-solver_cp-par	59.365
Solver	izplus-par	58.216
Solver	pumpkin-free	57.681
Solver	choco-solver_cp-sat-par	56.896
Solver	gencode-par	54.283
Solver	cp_optimizer-par	53.877
Solver	gencode_dexter-open	47.304
Solver	or-tools_cp-sat_ls-par	46.292
Solver	jacop-free	44.373
Solver	sicstus_prolog-free	43.548
Solver	yuck-par	38.321
Solver	scip-par	36.675
Solver	highs-par	33.598
Solver	cbc-par	26.334
Solver	atlantis-free	1.555

Table 2.25: Table displaying all the different LLM variants results from parallel-solver evaluation, “Total Score” is calculated as “Parallel Score” in Table 2.3, and compared to all of the single solvers in open category of the MiniZinc Challenge [17]

Type	Solver	TotalScore
LLM Variant	openai/gpt-oss-120b (Solvers Description + Problem Description)	77.260
LLM Variant	moonshotai/kimi-k2-instruct-0905 (Solvers Description + Problem Description)	76.964
LLM Variant	moonshotai/kimi-k2-instruct-0905 (Solvers Description)	76.964
LLM Variant	moonshotai/kimi-k2-instruct (Solvers Description)	76.964
Solver	or-tools_cp-sat-free	76.964
LLM Variant	openai/gpt-oss-120b (Simple)	74.488
LLM Variant	moonshotai/kimi-k2-instruct (Solvers Description + Problem Description)	74.464
Solver	chuffed-free	74.456
LLM Variant	gemini-2.5-flash (Solvers Description + Problem Description)	73.551
LLM Variant	moonshotai/kimi-k2-instruct-0905 (Problem Description)	72.605
LLM Variant	gemini-2.5-flash (Solvers Description)	71.686
LLM Variant	moonshotai/kimi-k2-instruct-0905 (Simple)	71.622
LLM Variant	openai/gpt-oss-120b (Solvers Description)	70.974
LLM Variant	moonshotai/kimi-k2-instruct (Simple)	70.939
Solver	picatsat-free	70.933
LLM Variant	openai/gpt-oss-120b (Problem Description)	70.740
LLM Variant	gemini-2.5-flash-lite (Simple)	70.145
LLM Variant	gemini-2.5-flash (Simple)	69.763
LLM Variant	gemini-2.5-flash-lite (Problem Description)	69.042
Solver	huub-free	68.497
LLM Variant	moonshotai/kimi-k2-instruct (Problem Description)	67.554
LLM Variant	gemini-2.5-flash-lite (Solvers Description + Problem Description)	63.062
Solver	choco-solver_cp-sat-free	60.808
Solver	izplus-free	58.672
Solver	pumpkin-free	58.543
Solver	choco-solver_cp-free	58.404
Solver	gurobi-free	55.384
LLM Variant	gemini-2.5-flash-lite (Solvers Description)	54.713
Solver	cp_optimizer-free	50.992
Solver	gecode-fd	50.073
LLM Variant	gemini-2.5-flash (Problem Description)	49.896
Solver	cplex-free	48.514
Solver	sicstus_prolog-free	44.592
Solver	jacop-free	44.549
Solver	or-tools_cp-sat_ls-free	43.222
Solver	scip-free	36.902
Solver	yuck-free	34.715
Solver	highs-free	34.418
Solver	cbc-free	22.615
Solver	atlantis-free	1.5

Table 2.26: Table displaying all the first LLM variants results given single-solver evaluation, “Total Score” is calculated as “Single Score” in Table 2.3, and compared to all of the single solvers in free category of the MiniZinc Challenge [17]

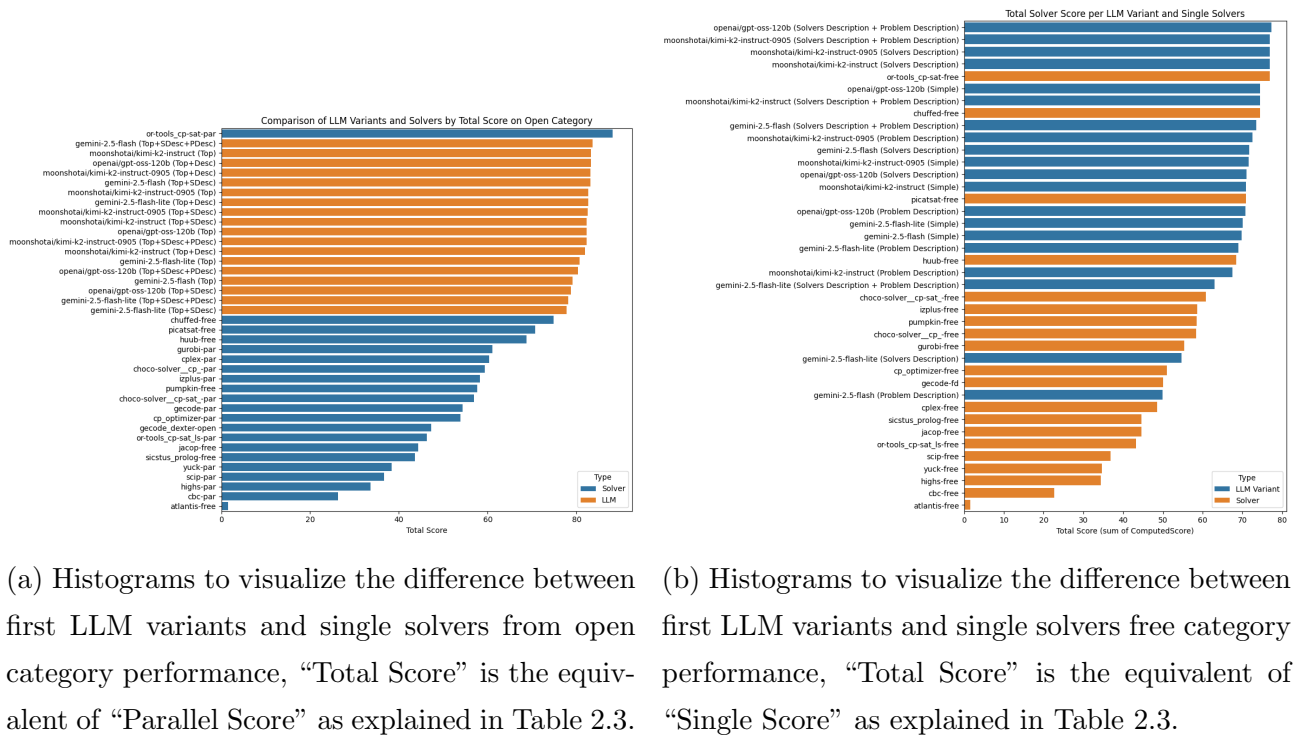


Figure 2.5: Histograms to visualize the difference between first LLM variants and single solvers

Category	Number	Description (summary)
Variables	27	Counts of variables (including constants, aliases, defined and introduced variables), ratios involving N_V and N_C , and statistics of domain size, degree, and domain/degree ratio.
Domains	18	Counts and ratios of variables by type (Boolean, integer, float, set) and constraints by type (Boolean, integer, float, set, array).
Constraints	27	Total N_C , ratios with N_V , annotation usage, and statistics of constraint domain, degree, and domain/degree ratio.
Global constraints	29	Total number and ratio of global constraints, plus counts per equivalence class of Gecode-supported globals.
Graphs	20	Statistics on structural properties of the constraint graph and variable graph (degree, clustering coefficient, diameter).
Solving	11	Information from the solve item, including labeled variables, goal type, and counts of search and heuristic annotations.
Objective	12	Domain, degree, and graph-related measures of the objective variable, including normalized and standardized forms relative to global domain and degree statistics.
Static total	144	Extracted from the FlatZinc model via syntactic and structural analysis.
Dynamic	11	Runtime indicators from short Gecode executions: solutions found, propagations, nodes, failures, search depth, memory usage, and timing measures for translation and feature extraction.

Table 2.27: The table shows a quick explanation of the features that can be extracted using the tool `mzn2feat` [25], divided per category, and counted.

Listing 2.1: Example output using `mzn2feat` [25] pretty print option on `EchoSched.mzn`, instance 14-10-0-2_1.

IDENTIFIER	VALUE	DESCRIPTION
=====		
<code>c_avg_deg_cons</code>	3.11367	Average of the constraints degree
<code>c_avg_dom_cons</code>	20.3934	Average of the constraints domain
<code>c_avg_domdeg_cons</code>	5.85385	Average of the ratio constraints domain/degree
<code>c_bounds_d</code>	0	No of constraints using 'boundsD' annotation
<code>c_bounds_r</code>	0	No of constraints using 'boundsR' annotation
<code>c_bounds_z</code>	0	No of constraints using 'boundsZ' or 'bounds' annotation
<code>c_cv_deg_cons</code>	0.963493	Coefficient of Variation of constraints degree
<code>c_cv_dom_cons</code>	1.50739	Coefficient of Variation of constraints domain
<code>c_cv_domdeg_cons</code>	0.566571	Coefficient of Variation of the ratio constraints domain/degree
<code>c_domain</code>	0	No of constraints using 'domain' annotation
<code>c_ent_deg_cons</code>	1.60699	Entropy of constraints degree
<code>c_ent_dom_cons</code>	5.54008	Entropy of constraints domain
<code>c_ent_domdeg_cons</code>	2.58764	Entropy of the ratio constraints domain/degree
<code>c_logprod_deg_cons</code>	6021.66	Logarithm of the product of constraints degree
<code>c_logprod_dom_cons</code>	14249.3	Logarithm of the product of constraints domain
<code>c_max_deg_cons</code>	141	Maximum of the constraints degree
<code>c_max_dom_cons</code>	1724.64	Maximum of the constraints domain
<code>c_max_domdeg_cons</code>	12.2315	Maximum of the ratio constraints domain/degree
<code>c_min_deg_cons</code>	1	Minimum of the constraints degree
<code>c_min_dom_cons</code>	1	Minimum of the constraints domain
<code>c_min_domdeg_cons</code>	1	Minimum of the ratio constraints domain/degree
<code>c_num_cons</code>	3906	Total no of constraints
<code>c_priority</code>	0	No of constraints using 'priority' annotation
<code>c_ratio_cons</code>	1.49426	Ratio no of constraints / no of variables
<code>c_sum_ari_cons</code>	13031	Sum of constraints arity
<code>c_sum_dom_cons</code>	79656.6	Sum of constraints domain
<code>c_sum_domdeg_cons</code>	22865.2	Sum of the ratio constraints domain/degree
<code>d_array_cons</code>	1	No of array constraints
<code>d_bool_cons</code>	910	No of boolean constraints
<code>d_bool_vars</code>	1820	No of boolean variables
<code>d_float_cons</code>	0	No of float constraints
<code>d_float_vars</code>	0	No of float variables
<code>d_int_cons</code>	2591	No of integer constraints
<code>d_int_vars</code>	794	No of integer variables
<code>d_ratio_array_cons</code>	0.000256016	Ratio array constraints / total no of constraints
<code>d_ratio_bool_cons</code>	0.232975	Ratio boolean constraints / total no of constraints
<code>d_ratio_bool_vars</code>	0.696251	Ratio boolean variables / total no of variables
<code>d_ratio_float_cons</code>	0	Ratio float constraints / total no of constraints
<code>d_ratio_float_vars</code>	0	Ratio float variables / total no of variables
<code>d_ratio_int_cons</code>	0.663338	Ratio integer constraints / total no of constraints
<code>d_ratio_int_vars</code>	0.303749	Ratio integer variables / total no of variables
<code>d_ratio_set_cons</code>	0	Ratio set constraints / total no of constraints
<code>d_ratio_set_vars</code>	0	Ratio set variables / total no of variables
<code>d_set_cons</code>	0	No of set constraints
<code>d_set_vars</code>	0	No of set variables
<code>gc_diff_globs</code>	1	No of different global constraints
<code>gc_global_cons</code>	404	Total no of global constraints
<code>gc_ratio_diff</code>	0.00247525	Ratio different global constraints / no of global constraints
<code>gc_ratio_globs</code>	0.103431	Ratio no of global constraints / total no of constraints
<code>o_deg</code>	1	Degree of the objective variable
<code>o_deg_avg</code>	0.214932	Ratio degree of the objective variable / average of var degree
<code>o_deg_cons</code>	0.000256016	Ratio degree of the objective variable / number of constraints
<code>o_deg_std</code>	-0.442948	Standardization of the degree of the objective variable
<code>o_dom</code>	6685	Domain size of the objective variable

o_dom_avg	12.79	Ratio domain of the objective variable / average of var domain
o_dom_deg	6685	Ratio domain of the objective variable / degree of the obj var
o_dom_std	4.13516	Standardization of the domain of the objective variable
s_bool_search	0	Number of 'bool_search' annotations
s_first_fail	1	Number of 'int_search' annotations
s_goal	2	Solve goal (1 = satisfy, 2 = minimize, 3 = maximize)
s_indomain_max	0	Number of 'indomain_max' annotations
s_indomain_min	1	Number of 'indomain_min' annotations
s_input_order	0	Number of 'input_order' annotations
s_int_search	1	Number of 'int_search' annotations
s_labeled_vars	1	Number of variables to be assigned
s_other_val	0	Number of other value search heuristics
s_other_var	0	Number of other variable search heuristics
s_set_search	0	Number of 'set_search' annotations
v_avg_deg_vars	4.65264	Average of the variables degree
v_avg_dom_vars	522.674	Average of the variables domain
v_avg_domdeg_vars	141.005	Average of the ratio variables domain/degree
v_cv_deg_vars	1.77237	Coefficient of Variation of variables degree
v_cv_dom_vars	2.85116	Coefficient of Variation of variables degree
v_cv_domdeg_vars	3.94885	Coefficient of Variation of the ratio variables domain/degree
v_def_vars	2334	Number of defined variables
v_ent_deg_vars	1.0029	Entropy of variables degree
v_ent_dom_vars	2.09955	Entropy of variables domain
v_ent_domdeg_vars	1.83264	Entropy of the ratio variables domain/degree
v_intro_vars	2753	Number of introduced variables
v_logprod_deg_vars	3665.47	Logarithm of the product of variables degree
v_logprod_dom_vars	6788.52	Logarithm of the product of variables domain
v_max_deg_vars	36	Maximum of the variables degree
v_max_dom_vars	6685	Maximum of the variables domain
v_max_domdeg_vars	6685	Maximum of the ratio variables domain/degree
v_min_deg_vars	0	Minimum of the variables degree
v_min_dom_vars	2	Minimum of the variables domain
v_min_domdeg_vars	0.1	Minimum of the ratio variables domain/degree
v_num_aliases	700	Number of alias variables
v_num_consts	10	Number of constant variables
v_num_vars	2614	Total no of variables variables
v_ratio_bounded	0.271614	Ratio (aliases + constants) / total no of variables
v_ratio_vars	0.669227	Ratio no of variables / no of constraints
v_sum_deg_vars	12162	Sum of variables degree
v_sum_dom_vars	1.36627e+06	Sum of variables domain
v_sum_domdeg_vars	368586	Sum of the ratio variables domain/degree

Bibliography

- [1] R. Barták, “Constraint Programming: In Pursuit of the Holy Grail”, Jan. 1999.
- [2] E. C. Freuder, “In Pursuit of the Holy Grail”, *Constraints*, vol. 2, no. 1, pp. 57-61, Apr. 1997, doi: <https://doi.org/10.1023/a:1009749006768>.
- [3] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack, ”MiniZinc: Towards a Standard CP Modelling Language,” Springer eBooks, pp. 529-543, Oct. 2007, doi: https://doi.org/10.1007/978-3-540-74970-7_38.
- [4] F. Chalumeau, I. Coulon, Q. Cappart, and L.-M. Rousseau, “SeaPearl: A Constraint Programming Solver guided by Reinforcement Learning”, arXiv.org, Apr. 20, 2021. <https://arxiv.org/abs/2102.09193>
- [5] R. Amadini, M. Gabbrielli, and J. Mauro, “Why CP Portfolio Solvers Are (under)Utilized? Issues and Challenges”, *Lecture Notes in Computer Science*, pp. 349-364, 2015, doi: https://doi.org/10.1007/978-3-319-27436-2_21.
- [6] Gomes, C.P., Selman, B.: “Algorithm portfolios”. *Artif. Intell.* 126(1-2), 43-62 (2001)
- [7] Rice, J.R.: “The algorithm selection problem”. *Adv. Comput.* 15, 65-118 (1976)
- [8] Kotthoff, L.: “Algorithm selection for combinatorial search problems: a survey”. *AI Mag.* 35(3), 48-60 (2014)
- [9] Roberto Amadini, Simone Gazza. From Portfolio Solvers to Agentic Solvers. LLM-Solve, Tias Guns; Serdar Kadioglu; Stefan Szeider; Dimos Tsouros, Aug 2025, Glasgow, United Kingdom, United Kingdom. pp.569 - 585, 10.5281/zenodo.17640236. hal-05446738
- [10] ”Gemini API Docs and Reference,” Google AI for Developers. <https://ai.google.dev/gemini-api/docs> (accessed Dec. 10, 2025).
- [11] ”API versions explained,” Google AI for Developers, 2025. <https://ai.google.dev/gemini-api/docs/api-versions> (accessed Dec. 11, 2025).
- [12] ”GroqCloud,” Groq.com, 2024. <https://console.groq.com/docs/overview> (accessed Dec. 10, 2025).

- [13] "Rate Limits - GroqDocs," GroqDocs, 2025. <https://console.groq.com/docs/rate-limits> (accessed Dec. 10, 2025).
- [14] "Rate limits," Google AI for Developers, 2025. <https://ai.google.dev/gemini-api/docs/rate-limits> (accessed Dec. 10, 2025).
- [15] Kelly Hong, Anton Troynikov, Jeff Huber, "Context Rot: How Increasing Input Tokens Impacts LLM Performance," Trychroma.com, 2025. <https://research.trychroma.com/context-rot?ref=blog.promptlayer.com> (accessed Dec. 11, 2025).
- [16] "MiniZinc - List of Problems and Globals used in the MiniZinc Challenge," Minizinc.org, 2025. <https://www.minizinc.org/challenge/globals/> (accessed Dec. 11, 2025).
- [17] "MiniZinc - Challenge 2025 Results," Minizinc.org, 2025. <https://www.minizinc.org/challenge/2025/results/> (accessed Dec. 11, 2025).
- [18] P. J. Stuckey, T. Feydy, A. Schutt, G. Tack, and J. Fischer, "The MiniZinc Challenge 2008-2013," *AI Magazine*, vol. 35, no. 2, p. 55, Jun. 2014, doi: <https://doi.org/10.1609/aimag.v35i2.2539>.
- [19] R. Amadini, Maurizio Gabbrielli, T. Liu, and J. Mauro, "On the Evaluation of (Meta-)solver Approaches," *Journal of Artificial Intelligence Research*, vol. 76, pp. 705-719, Mar. 2023, doi: <https://doi.org/10.1613/jair.1.14102>.
- [20] R. Amadini, Maurizio Gabbrielli, and J. Mauro, "Portfolio Approaches for Constraint Optimization Problems," *Lecture notes in computer science*, pp. 21-35, Jan. 2014, doi: https://doi.org/10.1007/978-3-319-09584-4_3.
- [21] Lindauer, M., van Rijn, J. N., & Kotthoff, L. (2019). "The algorithm selection competitions" 2015 and 2017. *Artificial Intelligence*, 272, 86-100.
- [22] Shi, F., Chen, X., Misra, K., Scales, N., Dohan, D., Chi, E., Schärli, N., and Zhou, D. (2023). Large Language Models Can Be Easily Distracted by Irrelevant Context. *arXiv preprint arXiv:2302.00093*. doi: <https://doi.org/10.48550/arXiv.2302.00093>
- [23] Sumanth P, "Agentic Prompt Engineering: A Deep Dive into LLM Roles and Role-Based Formatting," Clarifai.com, Jul. 2025. https://www.clarifai.com/blog/agentic-prompt-engineering#title_1 (accessed Jan. 03, 2026).
- [24] X. Lin et al., "LLM-based Agents Suffer from Hallucinations: A Survey of Taxonomy, Methods, and Directions", *arXiv (Cornell University)*, Sep. 2025, doi: <https://doi.org/10.48550/arxiv.2509.18970>.

- [25] R. Amadini, M. Gabbrielli, and J. Mauro. An Enhanced Features Extractor for a Portfolio of Constraint Solvers. In SAC, 2014.
- [26] R. Amadini, and Peter J. Stuckey. Sequential Time Splitting and Bounds Communication for a Portfolio of Optimization Solvers. In CP, 2014.
- [27] R. Amadini, M. Gabbrielli, and J. Mauro. SUNNY-CP: a Sequential CP Portfolio Solver. In SAC, 2015.
- [28] R. Amadini, M. Gabbrielli, and J. Mauro. A Multicore Tool for Constraint Solving. In IJCAI, 2015.
- [29] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, “A learning algorithm for boltzmann machines”, *Cognitive Science*, vol. 9, no. 1, pp. 147-169, Jan. 1985, doi: [https://doi.org/10.1016/S0364-0213\(85\)80012-4](https://doi.org/10.1016/S0364-0213(85)80012-4).
- [30] G. Hinton, O. Vinyals, and J. Dean, “Distilling the Knowledge in a Neural Network”, *arXiv.org*, Mar. 09, 2015. <http://arxiv.org/abs/1503.02531>
- [31] P.-H. Wang et al., “Contextual Temperature for Language Modeling”, *arXiv (Cornell University)*, Jan. 2020, doi: <https://doi.org/10.48550/arxiv.2012.13575>.
- [32] R. Wang, H. Wang, F. Mi, Y. Chen, R. Xu, and K.-F. Wong, “Self-Critique Prompting with Large Language Models for Inductive Instructions”, *arXiv.org*, May 23, 2023. <https://arxiv.org/abs/2305.13733> (accessed Mar. 05, 2024).
- [33] “Prompt Basics - GroqDocs”, *GroqDocs*, 2026. <https://console.groq.com/docs/prompting>

Acknowledgements

Here you can thank whoever you want.