This is the Dedication:

you can write whatever you want here,

or nothing at all …

Introduction

This is the introduction

Contents

List of Figures

List of Tables

Chapter 1

Experimental Evaluation

1.1 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 the make automatic testing possible.

1.1.1 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`[1], offered by Google DeepMind. Gemini is a family of large language models with multiple sizes and capabilities. This provider 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[2] was prefered as it is more stable and our only concern is its text generation capability.

- `Groq API`[3], 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 Section 1.1.1 and Section 1.1.1 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 1.1: Rate limits - Groq models[4]:
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 1.2: Rate limits - Gemini models[5]
This table shows all the offered models from Gemini API, in leftmost column and each relative rate limit.

1.1.2 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.

1.1.3 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"[6] - 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:

$$[\text{"}1^{st}\text{Solver"},\ \text{"}2^{nd}\text{Solver"},\ \text{"}3^{rd}\text{Solver"}]$$

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 1.1.5.

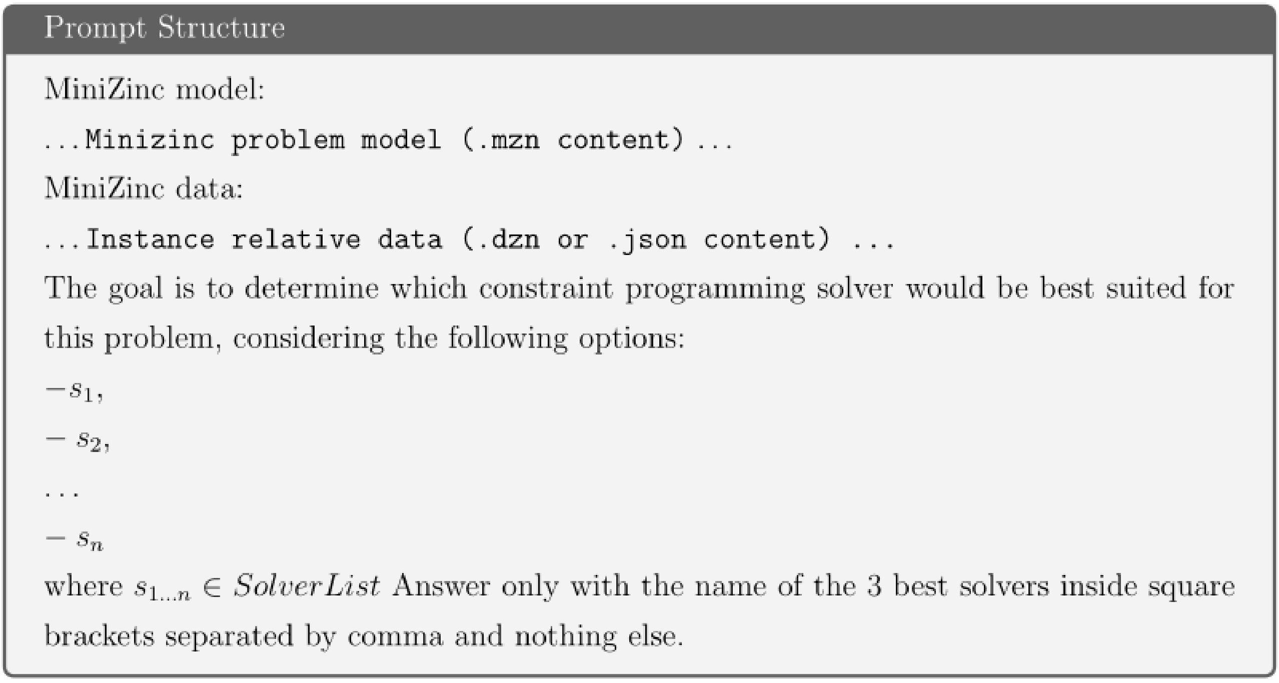
After all of this considerations, the resulting prompt structure is the one displayed in Figure 1.1

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

Figure 1.1: 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*[9][7][8]. 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[10] 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.

1.1.5 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 $(\mathcal{I}_1, \mathcal{S}_1, \tau_1)$, $(\mathcal{I}_2, \mathcal{S}_2, \tau_2)$, ..., 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

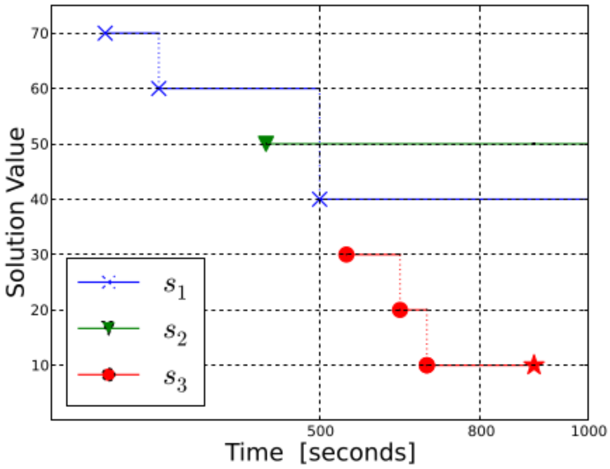


Figure 1.2: Solver performances example

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 $\text{score}_{\alpha,\beta}$[11][12] defined as follows:

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

Here, MIN(i) and 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 1.2 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*[11] as the evaluation metric. Which is a relative and meta-solver-specific measure, adopted in the 2015 ICON and 2017 OASC [13] 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 $(\mathcal{I}, S, \tau)$ be a scenario and

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

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

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

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.

1.1.6 Experiment Setup

We have defined both the structure of the queries posed to the LLMs (Section 1.1.4) and the way in which these queries are formulated (Section 1.1.3). 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 1.1.

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 `%`[10]), 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[14].

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

$$[e_1, e_2, \ldots, e_{30}\texttt{// array too long to display, dimensions: (150)}]$$

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-120
   b‘ in organization ‘org_01k9qqesvte4d9h5jnhmzvbmy4‘ 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_01k9qqesvte4d9h5jnhmzvbmy4‘
   service tier ‘on_demand‘ on tokens per day (TPD): Limit
   200000, Used 193047, Requested 10632. Please try again in 26
   m29.328s. 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.

1.2 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 descrption . . .

1.2.1 Preliminary tests

First most simple test to be done is made using raw scripts, so giving the .mzn model with relative instance data (either in .dzn or .json) to each of the available LLMs using the previously defined prompt structure Section 1.1.3.

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

Table 1.3: Initial tests giving plain scripts to all the LLMs, evaluated on parallel score

As shown in Section 1.2.1 and Table 1.4 the results aren't flattering and for most LLMs are actually quite bad, aside from the primitive formulation of the requests, a huge limitation is due to the rate limits,which doesn't allow many problems to be processed entirely. Those limitations lead to both the use of techniques emplying script manipulation, as previously explained in Section 1.1.6, and the choice to test only on the five best performing LLMs from now on.

1.2.2 Single Request Experiments

After the necessary steps to realize working experiments, we need to test on the 100 instances again, the experiments are made using a "single request setup"(ref funzione groq e gemini), where we send the prompt and recieve back the answer without keeping an history or previous context in any instance.

provider	model	LLM_Top1_TotalScore	InstancesCovered	LLM_Top1_AvgScore
gemini	gemini-2.5-flash-lite	64.363	85	0.794
gemini	gemini-2.5-flash	60.962	85	0.734
groq	moonshotai/kimi-k2-instruct-0905	59.680	75	0.817
groq	moonshotai/kimi-k2-instruct	58.609	75	0.837
groq	openai/gpt-oss-120b	58.166	74	0.796
groq	openai/gpt-oss-20b	57.154	75	0.828
groq	meta-llama/llama-4-maverick-17b-128e-instruct	56.297	72	0.804
groq	meta-llama/llama-4-scout-17b-16e-instruct	54.305	69	0.798
gemini	gemini-2.0-flash	43.412	67	0.700
groq	qwen/qwen3-32b	42.116	65	0.779
gemini	gemini-2.5-pro	37.640	51	0.738
groq	llama-3.1-8b-instant	36.082	72	0.546
groq	groq/compound-mini	25.422	35	0.726
groq	llama-3.3-70b-versatile	7.454	10	0.745
groq	groq/compound	5.197	9	0.577
groq	allam-2-7b	0.0	4	0.0

Table 1.4: Initial tests giving plain scripts to all the LLMs, evaluated on single best score

Simple Setup

First of all it is due to test the the LLMs on the same setup of preliminary experiments, so only using raw scripts.

provider	model	LLM_TotalScore	InstancesCovered	LLM_AvgScore
gemini	gemini-2.5-flash	79.105	100	0.791
gemini	gemini-2.5-flash-lite	80.740	100	0.807
groq	moonshotai/kimi-k2-instruct	83.268	100	0.832
groq	moonshotai/kimi-k2-instruct-0905	82.656	100	0.826
groq	openai/gpt-oss-120b	82.226	100	0.822

Table 1.5: Tests on sanitized scripts given to the 5 best performing LLMs, evaluated on parallel score

As shown in the tables, the results are pretty good in parallel evaluation, in fact all of the LLMs are putting up better score the all of the LLMs beside `or-tools_cp-sat-par`, so the SBS for open category, that, anyway is winning with a wide margin from both the tested LLMs and the other single solvers.

provider	model	LLM_Top1_TotalScore	InstancesCovered	LLM_Top1_AvgScore
groq	openai/gpt-oss-120b	74.488	100	0.744
groq	moonshotai/kimi-k2-instruct-0905	71.622	100	0.753
groq	moonshotai/kimi-k2-instruct	70.939	100	0.723
gemini	gemini-2.5-flash-lite	70.145	100	0.738
gemini	gemini-2.5-flash	69.763	98	0.742

Table 1.6: Tests on sanitized scripts given to the 5 best performing LLMs, evaluated on single best score

In the tables we can see, there is more variation on the score distribution, in fact the only LLM placing itself above all the single solvers, beside the SBS for free category (`or-tools_cp-sat-free`), is `gpt-oss-120b`, the other LLMs are still giving considerably good results against other single solvers, but there is still a big margin for improvement, As can be directly seen from the closed gap table Table 1.7.

provider	model	InstancesCovered	AS	SBS	VBS	ClosedGap
groq	openai/gpt-oss-120b	100	74.488	76.964	89.0	-0.205
groq	moonshotai/kimi-k2-instruct-0905	100	71.622	76.964	89.0	-0.443
groq	moonshotai/kimi-k2-instruct	100	70.939	76.964	89.0	-0.500
gemini	gemini-2.5-flash-lite	100	70.145	76.964	89.0	-0.566
gemini	gemini-2.5-flash	98	69.763	76.964	89.0	-0.598

Table 1.7: Tests on sanitized scripts given to the 5 best performing LLMs, closed gap

Problem Description

After these first tests, it becomes clear that it is needed a way to improve the Agentic Solver performance. What seems the easiest solution to this problem, is simply to give the LLM more context information. But, as previously stated, too big of a context might actually ”distract” the LLM and even worsen its performance[6][14], instead of making it better. This conflict, raises a new question, it is needed that we understand which information are actually more understandable for the LLMs, and to do that we need to test on different setups. Starting by giving our AS the problem description (PD), a brief text relative to a single `.mzn` describing the content of the problem. The descriptions were automatically generated via another LLM (`GPT-5.1`), and later slightly modified by hand to fix minor errors. The PDs are then incorporated in the prompt structure (Figure 1.1) as follows:

$$\text{"Problem Description: "} + PD + PromptStructure$$

As shown in (ref a tabella) adding these context information gave opposite results on parallel and single execution. Giving consistently worse results on single solver answers, and giving slightly better results for parallel executions with better performance on 3 out of the 5 tested LLMs.

As can be deduced from the results, sadly, the closed gap is still negative in this case.

1.2.3 Multi-turn experiments

From previously explained experiments, it turn out that the context information we are giving are either limited or not helping, in a broader sense. The obvious follow-up steps are to test again, by giving different context information, this anyway raises a couple questions. First of all, we are already reaching the maximum limit of requested tokens (TPM) in a single

provider	model	LLM_TotalScore	InstancesCovered	LLM_AvgScore
gemini	gemini-2.5-flash	59.154	75	0.788
gemini	gemini-2.5-flash-lite	82.620	100	0.826
groq	moonshotai/kimi-k2-instruct	81.889	100	0.818
groq	moonshotai/kimi-k2-instruct-0905	83.182	100	0.831
groq	openai/gpt-oss-120b	83.249	100	0.832

Table 1.8: Test on sanitized scripts combined with textual problem description, parallel score evaluation

provider	model	LLM_Top1_TotalScore	InstancesCovered	LLM_Top1_AvgScore
groq	moonshotai/kimi-k2-instruct-0905	72.605	100	0.748
groq	openai/gpt-oss-120b	70.740	100	0.721
gemini	gemini-2.5-flash-lite	69.042	100	0.719
groq	moonshotai/kimi-k2-instruct	67.554	100	0.718
gemini	gemini-2.5-flash	49.896	73	0.723

Table 1.9: Test on sanitized scripts combined with textual problem description, single best evaluation

provider	model	InstancesCovered	AS	SBS	VBS	ClosedGap
groq	moonshotai/kimi-k2-instruct-0905	100	72.605	76.964	89.0	-0.362
groq	openai/gpt-oss-120b	100	70.740	76.964	89.0	-0.517
gemini	gemini-2.5-flash-lite	100	69.042	76.964	89.0	-0.658
groq	moonshotai/kimi-k2-instruct	100	67.554	76.964	89.0	-0.781
gemini	gemini-2.5-flash	73	49.896	76.964	89.0	-2.248

Table 1.10: Test on sanitized scripts combined with textual problem description, closed gap

request setup, so adding more information, while keeping the the same experiment configuration, wouldn't be a viable option. Another problem with the previous setup is that we are actively wasting tokens, by giving the LLMs redundant context. To further explain this, for each problem, we have 5 different instances we are testing on, so 5 different sets of data, while the problem stays the same, and clearly it's the same thing for the problem description as well. In our context of limited resources, it is imperative we solve those issues, and to do that we switched to a multi-turn experiment setup (or chat-like setup).

Setup Explanation

The main idea is to split the conversation with the use of a technique called role-based formatting[15]. To unlock the full potential of LLMs, especially in multi-turn setups, the model must understand who is speaking, what role they are playing, and what has already happened in the conversation. This is where role come in, such as `system`, `user`, and `assistant`, which define the context and intent behind every message. In particular, `system` help us define the general instructions for the conversation. In our case, we are using that by giving the LLM what were the redundant information: `.mzn` content, verbal problem descriptions and answer instructions. Then we used `user` which is used in the questions or commands that the model responds to. In our case,that contains the instance relative data (`.dzn` or `.json` content). Allowing us to further refine the conversation, giving clear instructions, while also limiting the token usage, and consequently enabling longer, more complex requests. This setup also enables longer instance data, by simply giving us the possibility to directly split messages, while the LLM is retaining information.

Solvers Description

With this improved setup we can add a new element to the context information: the solvers description. We generated a description of each of the involved solvers, and than gave it to the LLM, in the system prompt giving it a better understanding of its options for a result. To reach a better understanding of information importance, we tried to add this information in two different ways: combined with all the previous context, and as the only text information combined with the `.mzn` model, and instance data.

provider	model	LLM_TotalScore	InstancesCovered	LLM_AvgScore
gemini	gemini-2.5-flash	83.137	100	0.831
gemini	gemini-2.5-flash-lite	77.746	100	0.777
groq	moonshotai/kimi-k2-instruct	82.236	100	0.822
groq	moonshotai/kimi-k2-instruct-0905	82.488	100	0.824
groq	openai/gpt-oss-120b	78.799	100	0.787

Table 1.11: Test with sanitized scripts combined with solvers description in a multi turn setup, parallel score evaluation

In the case of parallel evaluation, adding only solver description haven't produced any better results, beside for `gemini-2.5-flash`, but still not reaching the open category SBS score.

provider	model	LLM_Top1_TotalScore	InstancesCovered	LLM_Top1_AvgScore
groq	moonshotai/kimi-k2-instruct	76.964	100	0.769
groq	moonshotai/kimi-k2-instruct-0905	76.964	100	0.769
gemni	gemini-2.5-flash	71.686	100	0.716
groq	openai/gpt-oss-120b	70.974	100	0.716
gemni	gemini-2.5-flash-lite	54.713	100	0.552

Table 1.12: Test with sanitized scripts combined with solvers description in a multi turn setup, single best score evaluation

On the other hand, talking about single solver evaluation, it showed great improvement for two particular models, `moonshotai/kimi-k2-instruct-0905` and `moonshotai/kimi-k2-instruct` which actually reached the SBS score, giving for the first time a non-negative closed gap, but a 0 one. Sadly, when looking into the answers, we discovered that the improvement is simply due to the LLMs always giving the same answer for the best solver, specifically `or-tools_cp-sat-free`, the SBS, invalidating the purpose of an LLM involvement.

provider	model	InstancesCovered	AS	SBS	VBS	ClosedGap
groq	moonshotai/kimi-k2-instruct	100	76.964	76.964	89.0	0.0
groq	moonshotai/kimi-k2-instruct-0905	100	76.964	76.964	89.0	0.0
gemini	gemini-2.5-flash	100	71.686	76.964	89.0	-0.438
groq	openai/gpt-oss-120b	100	70.974	76.964	89.0	-0.497
gemini	gemini-2.5-flash-lite	100	54.713	76.964	89.0	-1.848

Table 1.13: Test with sanitized scripts combined with solvers description in a multi turn setup, closed gap

Solver Description and Problem Description

As previously stated, the next step was to try using both textual information, so both solvers description and problem description together. With this setup, we reached the maximum ammount of information given as context to the LLMs.

provider	model	LLM_TotalScore	InstancesCovered	LLM_AvgScore
gemini	gemini-2.5-flash	83.650	100	0.836
gemini	gemini-2.5-flash-lite	78.147	100	0.781
groq	moonshotai/kimi-k2-instruct	80.417	99	0.812
groq	moonshotai/kimi-k2-instruct-0905	82.218	100	0.822
groq	openai/gpt-oss-120b	80.295	100	0.802

Table 1.14: Test with sanitized scripts combined with both solvers description, and problem description in a multi turn setup, parallel score evaluation

In the case of parallel evaluation, while we reached the best score yet, with `gemini-2.5-flash`, the results aren't improving for other LLMs and we are still far back from the open category SBS.

provider	model	LLM_Top1_TotalScore	InstancesCovered	LLM_Top1_AvgScore
groq	openai/gpt-oss-120b	77.260	100	0.780
groq	moonshotai/kimi-k2-instruct-0905	76.964	100	0.769
groq	moonshotai/kimi-k2-instruct	74.464	99	0.752
gemini	gemini-2.5-flash	73.551	100	0.750
gemini	gemini-2.5-flash-lite	63.062	100	0.630

Table 1.15: Test with sanitized scripts combined with both solvers description, and problem description in a multi turn setup, single best score evaluation

In the case of single solver evaluation, while `moonshotai/kimi-k2-instruct-0905` still is giving the SBS as the only answer. We reached better results for 2 LLMs (`gemini-2.5-flash`), and `gpt-oss-120b`, which finally gives us the first positive closed gap yep, surpassing `or-tools_cp-sat-free`

provider	model	InstancesCovered	AS	SBS	VBS	ClosedGap
groq	openai/gpt-oss-120b	100	77.260	76.964	89.0	0.024
groq	moonshotai/kimi-k2-instruct-0905	100	76.964	76.964	89.0	0.0
groq	moonshotai/kimi-k2-instruct	99	74.464	76.964	89.0	-0.207
gemini	gemini-2.5-flash	100	73.551	76.964	89.0	-0.283
gemini	gemini-2.5-flash-lite	100	63.062	76.964	89.0	-1.155

Table 1.16: Test with sanitized scripts combined with both solvers description, and problem description in a multi turn setup, closed gap

1.2.4 Features

Bibliography

[1] ”Gemini API Docs and Reference,” Google AI for Developers. https://ai.google.dev/gemini-api/docs (accessed Dec. 10, 2025).

[2] ”API versions explained,” Google AI for Developers, 2025. https://ai.google.dev/gemini-api/docs/api-versions (accessed Dec. 11, 2025).

[3] ”GroqCloud,” Groq.com, 2024. https://console.groq.com/docs/overview (accessed Dec. 10, 2025).

[4] ”Rate Limits - GroqDocs,” GroqDocs, 2025. https://console.groq.com/docs/rate-limits (accessed Dec. 10, 2025).

[5] ”Rate limits,” Google AI for Developers, 2025. https://ai.google.dev/gemini-api/docs/rate-limits (accessed Dec. 10, 2025).

[6] 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).

[7] ”MiniZinc - List of Problems and Globals used in the MiniZinc Challenge,” Minizinc.org, 2025. https://www.minizinc.org/challenge/globals/ (accessed Dec. 11, 2025).

[8] ”MiniZinc - Challenge 2025 Results,” Minizinc.org, 2025. https://www.minizinc.org/challenge/2025/results/ (accessed Dec. 11, 2025).

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] Lindauer, M., van Rijn, J. N., & Kotthoff, L. (2019). "The algorithm selection competitions" 2015 and 2017.Artificial Intelligence,272, 86-100.

[14] 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

[15] 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).

[16] R. Amadini, M. Gabbrielli, and J. Mauro. An Enhanced Features Extractor for a Portfolio of Constraint Solvers. In SAC, 2014.

Acknowledgements

Here you can thank whoever you want.