0.1 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.

0.1.1 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`[**?**], 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[**?**] was prefered as it is more stable and our only concern is its text generation capability.

- `Groq API`[**?**], 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 0.1.1 and Section 0.1.1 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: Rate limits - Groq models[**?**]:
This table shows all the offered models from Groq API, in leftmost column and each relative rate limit.

0.1.2 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

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: Rate limits - Gemini models[**?**]
This table shows all the offered models from Gemini API, in leftmost column and each relative rate limit.

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.

0.1.3 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"[**?**] - 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 0.1.5.

After all of this considerations, the resulting prompt structure is the one displayed in Figure 1

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: Example of prompt

0.1.4 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.

- 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*[?][?][?]. 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[?] 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.

0.1.5 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

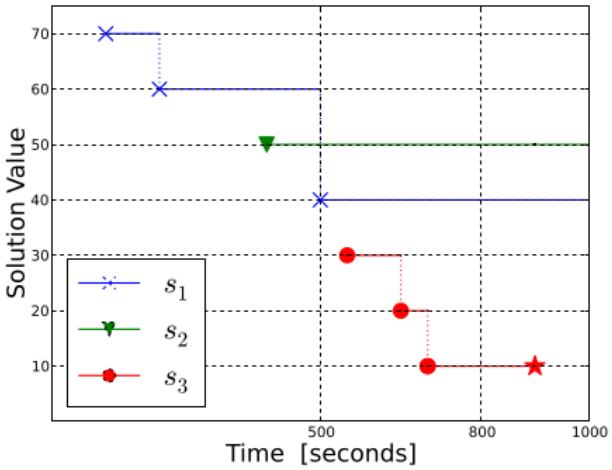


Figure 2: Solver performances example

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 $\text{score}_{\alpha,\beta}$[**?**][**?**] 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, $\text{MIN}(i)$ 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 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*[**?**] as the evaluation metric. Which is a relative and meta-solver-specific measure, adopted in the 2015 ICON and 2017 OASC [**?**] 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

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

and

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

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

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

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

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

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

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

The assumption $m_{\mathrm{VBS}} > m_{\mathrm{SBS}}$ 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.

0.1.6 Experiment Setup

We have defined both the structure of the queries posed to the LLMs (Section 0.1.4) and the way in which these queries are formulated (Section 0.1.3). 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.

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 `%[?`]), 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[**?**].

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.

0.2 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 ...

0.2.1 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 0.1.3.

As reported in **??** for parallel-solver evaluation, and in **??** for single-solver evaluation, the scores are under 70, meaning a lower performance than 3 of the single solvers. 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

Model	Single Score	Parallel Score	Closed Gap
gemini-2.5-flash-lite	64.363418	69.040165	-1.046972
gemini-2.5-flash	60.962014	69.426461	-1.329583
moonshotai/kimi-k2-instruct-0905	59.680205	66.186425	-1.436084
moonshotai/kimi-k2-instruct	58.609073	65.816023	-1.525081
openai/gpt-oss-120b	58.166305	64.508145	-1.561869
openai/gpt-oss-20b	57.154105	63.328830	-1.645969
meta-llama/llama-4-maverick-17b-128e-instruct	56.297161	63.548809	-1.717170
meta-llama/llama-4-scout-17b-16e-instruct	54.305367	57.814521	-1.882661
gemini-2.0-flash	43.412659	53.748095	-2.787700
qwen/qwen3-32b	42.116501	48.018123	-2.895394
gemini-2.5-pro	37.640862	41.594014	-3.267260
llama-3.1-8b-instant	36.082076	56.228369	-3.396774
groq/compound-mini	25.422973	29.623209	-4.282403
llama-3.3-70b-versatile	7.454830	9.496286	-5.775317
groq/compound	5.197411	8.246286	-5.962878

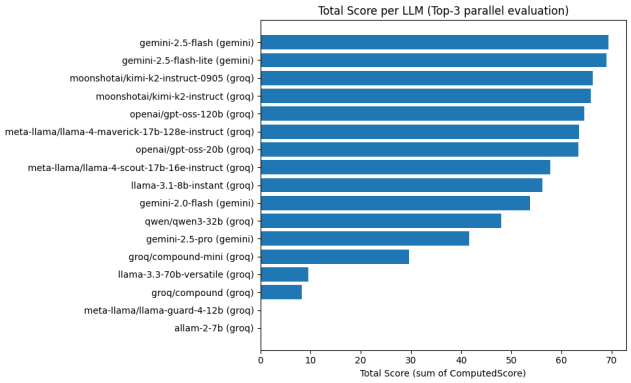
Table 3: Initial tests giving plain scripts to all the LLMs, In this table:column "Model" contains the names of each tested LLM," Total Score" represents the sum of the score reached in every instance, in this table "Total Score" represents the sum of the score reached in every instance, score was calculated 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 0.1.5

(showed in Section 0.1.1 and Section 0.1.1). This problem is predominant in problems with large instance data, such as: `ihtc-2024-marte` or `gt-sort`

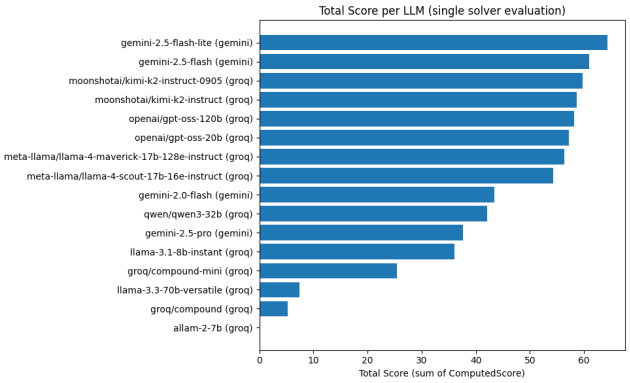
These constraints motivated both the adoption of script manipulation techniques, as described in Section 0.1.6, and, simple time issues due to tests taking even up to 48 hours, the decision to restrict subsequent experiments to the five best-performing LLMs identified in this preliminary phase, namely: `gpt-oss-120b`, `gemini-2,5-flash`, `gemini-2,5-flash-lite`, `kimi-k2-instruct-0905` and `kimi-k2-instruct`.

0.2.2 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



(a) Scores obtained by all of the considered LLM from parallel-solver evaluation



(b) Scores obtained by all of the considered LLM from single-solver evaluation

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	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 4: Tests on sanitized scripts given to the 5 best performing LLMs, parallel-solver evaluation

As shown in Table 4, 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, as will be later reported in Table 16 and Figure 5.

Greater variability emerges in the single-solver evaluation (Table 5). 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 17 and Figure 6. 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 reported in Table 6.

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 5: Tests on sanitized scripts given to the 5 best performing LLMs, single-solver evaluation

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

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[**? ?**]. 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 1) as:

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 $s_1 \dots s_n \in SolverList$.

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

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 7: Test on sanitized scripts combined with textual problem description, parallel-solver evaluation

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 8: Test on sanitized scripts combined with textual problem description, single-solver evaluation

As shown in Table 7 and Table 8, 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.

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

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

0.2.3 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 [**?**]. 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.

In the parallel-solver evaluation (Table 10), 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.

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 10: Test with sanitized scripts combined with solvers description in a multi turn setup, parallel-solver evaluation

On the other hand, the effects are more pronounced in the single-solver evaluation, displayed in Table 11. 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

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 11: Test with sanitized scripts combined with solvers description in a multi turn setup, single-solver evaluation

decision-making role of the LLM, thereby undermining the intended purpose of employing an LLM in the first place.

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

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	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 13: Test with sanitized scripts combined with both solvers description, and problem description in a multi turn setup, parallel-solver evaluation

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.

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 14: Test with sanitized scripts combined with both solvers description, and problem description in a multi turn setup, single-solver evaluation

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`.

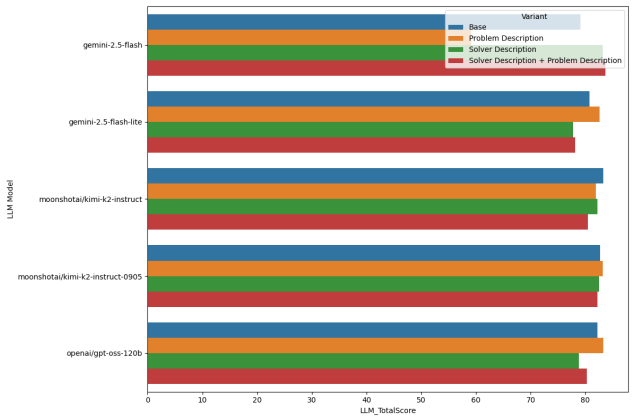
Model	Instances Covered	AS	SBS	VBS	Closed Gap
openai/gpt-oss-120b	100	77.260	76.964	89.0	0.024
moonshotai/kimi-k2-instruct-0905	100	76.964	76.964	89.0	0.0
moonshotai/kimi-k2-instruct	99	74.464	76.964	89.0	-0.207
gemini-2.5-flash	100	73.551	76.964	89.0	-0.283
gemini-2.5-flash-lite	100	63.062	76.964	89.0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

Table 15: Test with sanitized scripts combined with both solvers description, and problem description in a multi turn setup, closed gap

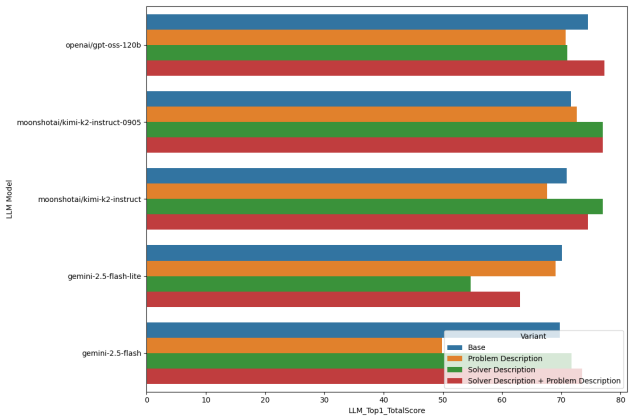
Basic Tests Evaluation

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

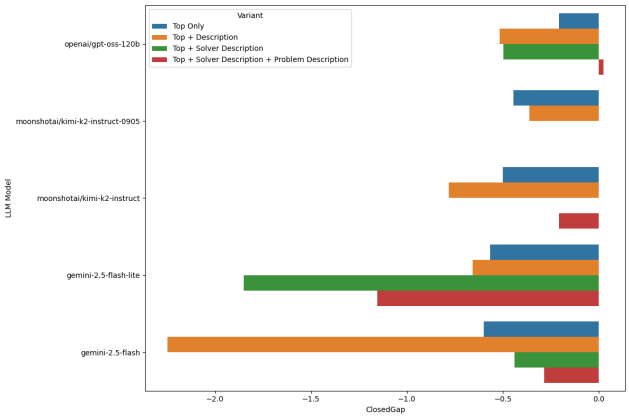
To better display single LLMs performances, all variants scores are put together against one another, using histograms for better visualization. In Figure 4a for parallel-solver evaluation, Figure 4b for single-solver evaluation and Figure 4c for closed gap evaluation.



(a) Histogram to display all variants performances in parallel-solver evaluation



(b) Histogram to display all variants performances in single-solver evaluation



(c) Histogram to display all variants performances in 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 16 and Figure 5 are shown the performance of all the configurations, scored with parallel-solvers evaluation, when put agains single solvers from the open category. On the other hand, looking at Table 17 and Figure 6, the resulting score of single-solver evaluation of all the variants is put against all the single solvers from free category.

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	gecode-par	54.283
Solver	cp_optimizer-par	53.877
Solver	gecode_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 16: Table displaying all the different LLM variants results from parallel-solver evaluation, and compared to all of the single solvers in open category of the MiniZinc Challenge[**?**]

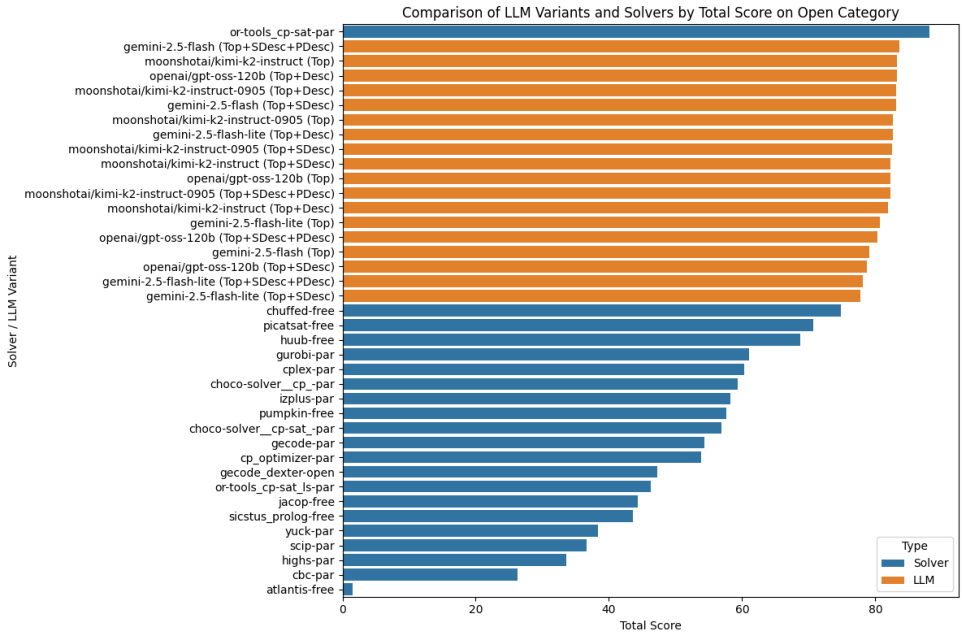


Figure 5: Histograms to visualize the difference between LLMs and single solvers from open category performance

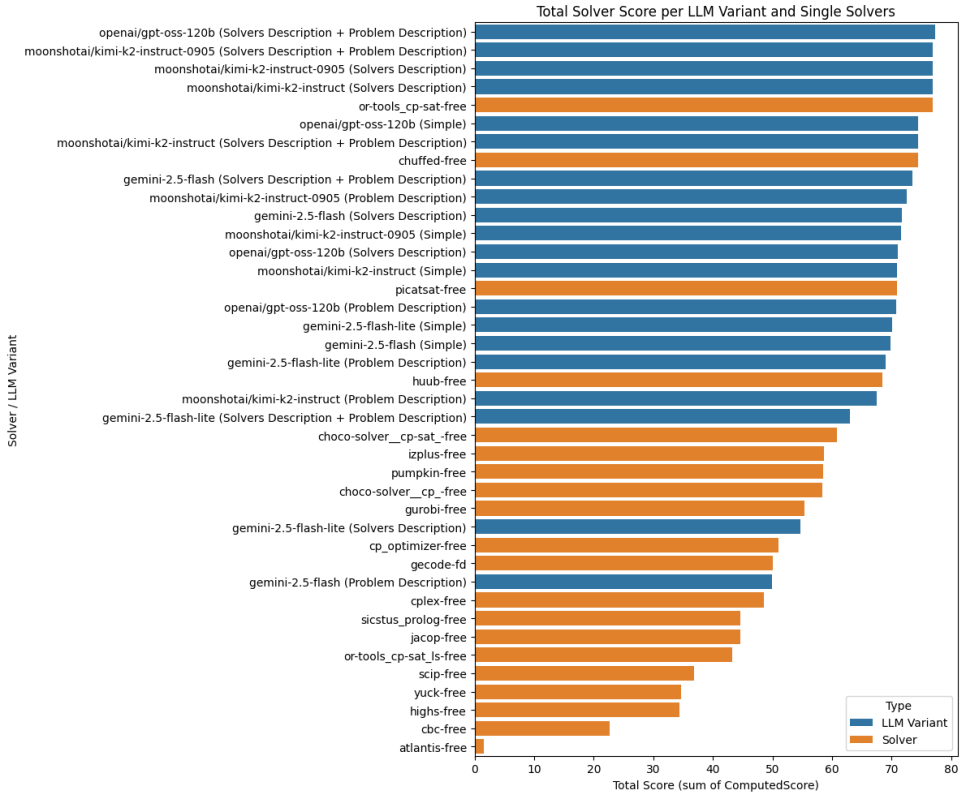


Figure 6: Histograms to visualize the difference between LLMs and single solvers from free category performance

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 17: Table displaying all the different LLM variants results given single-solver evaluation, and compared to all of the single solvers in free category of the MiniZinc Challenge[**?**]