This chapter presents the initial experimental study aimed at assessing the potential of LLMs for algorithm selection and identifying effective prompting strategies and contextual representations.

The first section reports results obtained with the most basic configuration, using only raw scripts and instance data, intended to estimate the baseline performance of all candidate LLMs. These results are used to select the five best-performing models for subsequent analysis. The second section examines refined prompt formulations based on sanitized scripts and the inclusion of problem descriptions, using a single-request setup in which each instance is handled independently.

The third section investigates multi-turn experiments, where prompts are enriched with solver descriptions and combined problem-solver descriptions. This setup leverages conversational state to reduce redundancy and mitigate rate-limit constraints inherent to single-request configurations. From this stage onward, experiments are conducted only with the best-performing model.

The fourth section evaluates the use of structured contextual representations derived from a feature extraction tool [**?** ], in combination with the prompt strategies developed earlier. Finally, the fifth section analyzes the effect of sampling temperature tuning [**?** ] by testing multiple temperature values on the five most effective configuration variants identified across the study.

## 0.1 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 **??**.

As reported in **??**, (and more in depth in **??**, **??** and **??**) 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 **??** and **??**). 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 **??**, 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`.

| 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 1: 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 **??**.

## 0.2 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.

### 0.2.1 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.

As shown in **??** and **??**, performance in the parallel-solver evaluation is generally strong. All tested LLMs outperform every individual solver except `or-tools_cp-sat-par`, which cor-

| 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: Tests on sanitized scripts given to the 5 best performing LLMs, columns content is calculated as in **??**.

responds 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 (**??**). In this case, only `gpt-oss-120b` consistently outperforms all individual solvers other than the SBS in the free category (`or-tools_cp-sat-f`
as will be later reported in **??** and **??**. 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 **??**.

### 0.2.2 Problem Description

The results of the baseline experiments indicate that further improvements 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 (**??**) 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 | 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 3: Test on sanitized scripts combined with textual problem description. Columns are calculated as in **??**.

As shown in **??**, 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.

## 0.3 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.

### 0.3.1 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.

### 0.3.2 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 Examples can be found in: . 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 LLMs.

| 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 4: Test with sanitized scripts combined with solvers description in a multi turn setup. Columns are calculated as in **??**.

In the parallel-solver evaluation (**??**), 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 **????**. 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.

### 0.3.3 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.

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,

| 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 5: Test with sanitized scripts combined with both solvers description, and problem description in a multi turn setup. Columns are calculated as in **??**.

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`.

### 0.3.4 Basic Tests Evaluation

In this initial testing phase, a positive closed gap has been achieved, showing an algorithm selection process with better performance than always choosing the single best solver. Moreover, when we put the performance respect to the "non-best solvers", these primitive configurations 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 **??** for parallel-solver evaluation, **??** for single-solver evaluation and **??** for closed gap evaluation.

To give results another point of view, the performance of each configuration is displayed against the single solvers of the corresponding category. In **??** and **??** 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 **??** and **??**, the resulting score of single-solver evaluation of all the variants is put against all the single solvers from free category.

## 0.4 Feature Extraction

In the last sections, we exposed the importance of textual information, and giving a richer context to the LLM instead of relying on its understanding of a `.mzn` script. Even though a positive 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 information 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 **??**, 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 [? ]and context rot [? ], due to the forced removal of elements in data, leaving incomplete arrays as input.

For these tasks we decided to employ a feature-extractor [? ], 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 [? ? ? ].

### 0.4.1 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 ?? for the description by category,

and to Listing ?? as an example output.

### 0.4.2 Testing and Results

To evaluate the effectiveness of feature-based representations, the pretty-printed `mzn2feat` output (e.g., ??) was incorporated into the prompt under several configurations:

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

Each configuration was further extended by incrementally adding the `.mzn` model and, subsequently, instance data (`.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 + .mzn + Problem Description | 71.916389 | 78.530605 | -0.419420 |
| Features + .mzn | 70.368365 | 78.371276 | -0.548041 |
| Features + .mzn + Instance Data | 68.422708 | 74.803213 | -0.709699 |
| Features + .mzn + Instance Data + Solver Description | 63.585459 | 66.346631 | -1.111610 |
| Features + .mzn + Instance Data + Problem Description | 62.516550 | 69.284744 | -1.200422 |
| Features + .mzn + Solver Description | 61.687990 | 66.874762 | -1.269264 |
| Features + .mzn + Solver Description + Problem Description | 58.576543 | 64.774159 | -1.527784 |
| Features + .mzn + Instance Data + Solver Description + Problem Description | 54.188182 | 59.410873 | -1.892398 |

Table 6: Test with all the possible combinations involving features extracted using `mzn2feat`, `mzn2feat`, 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 ??.

Results are reported in ??. Configurations combining feature representations with textual descriptions consistently achieve higher scores than those including raw `.mzn` scripts or instance data. The addition of full model code or data is associated with systematic score reductions across both single and parallel evaluations, most probably because of some problems explained earlier such as context rot [? ].

The highest-performing configurations in this group are those combining features with solver and/or problem descriptions. Although these variants do not exceed the best-performing configuration identified in earlier experiments, the top three feature-based setups achieve higher scores than the second-best configuration among the variants proposed in ??.

## 0.5 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 [**?** ].

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 $\tau$ 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. 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 [**? ?** ].

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.

### 0.5.1 Choosing Sampling Temperatures

In practical applications, lower temperatures are typically associated with tasks emphasizing consistency and structural correctness, whereas higher temperatures are used in contexts where output diversity is beneficial [**?** ]. Increased stochasticity, however, can also raise the likelihood of incoherent or factually incorrect outputs [**?** ]. Temperature selection therefore constitutes a trade-off between determinism and diversity.

An ablation study on temperature was conducted only for `gpt-oss-120b`, the model accessed through the Groq API [**?** ]. The tested values were based on the preset recommendations in the provider documentation [**?** ] (Table **??**).

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

Table 7: Temperature setting suggestions from Groq documentation [**?** ]. In the leftmost column is presented the basic scenario in wich 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.

### 0.5.2 Testing and Results

As anticipated, the tests were all made only using `gpt-oss-120b` for LLM. Due to rate-limit constraints, temperature tuning experiments was restricted to the five best-performing configuration variants identified earlier, which we will later refer to using this numbers:

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.

4. Features extracted through `mzn2feat` and text description for problems.

5. `.mzn` scripts and instance data

As displayed in **??** and in **??**, in parallel-solver evaluation, temperature variation produces only moderate changes. Although some configurations yield higher parallel scores than their counterparts with no temperature configuration, all remain below the performance of `or-tools_cp-sat-par` (displayed in **??**).

In the single-solver evaluation (**??**), temperature has a more pronounced effect. Configurations based on raw scripts show limited sensitivity to temperature, with only marginal improvements with configuration 5, and a decrease in scores for configuration 1. In contrast, feature-based configurations exhibit clearer trends: temperatures in the range 0.2-0.3 are associated with improved single-solver scores. Notably, configuration 3 achieves positive closed-gap values at $\tau$=0.3 and $\tau$=0.7, displayed in **??**, indicating performance above the SBS baseline under those settings, displaying highest scores yet.

| Variant | Temperature | Single Score | Parallel Score | Closed Gap |
|---|---|---|---|---|
| **Features + Solvers Descripton** | **0.3** | **78.032** | 82.873 | **0.089** |
| **Features + Solvers Descripton** | **0.7** | **77.896** | 83.767 | **0.077** |
| Scripts + Solvers Descripton + Problem Description | 0.0 | 76.971 | 82.042 | 0.001 |
| Features + Solvers Descripton + Problem Description | 0.3 | 76.529 | 82.959 | -0.036 |
| Features + Solvers Descripton | 0.2 | 76.326 | 84.044 | -0.053 |
| Features + Problem Description | 0.2 | 76.298 | 83.363 | -0.055 |
| Features + Solvers Descripton + Problem Description | 0.2 | 76.156 | 83.763 | -0.067 |
| Features + Solvers Descripton + Problem Description | 0.0 | 75.860 | 83.146 | -0.092 |
| Features + Problem Description | 0.8 | 75.676 | 83.009 | -0.107 |
| Features + Solvers Descripton | 0.0 | 75.477 | 83.444 | -0.124 |
| Features + Solvers Descripton + Problem Description | 0.8 | 74.909 | 82.419 | -0.171 |
| Scripts + Solvers Descripton + Problem Description | 0.3 | 74.907 | 80.042 | -0.171 |
| Features + Solvers Descripton | 0.8 | 74.464 | 84.421 | -0.208 |
| Scripts | 0.8 | 74.460 | 83.222 | -0.208 |
| Scripts | 0.2 | 74.456 | 83.119 | -0.208 |
| Scripts | 0.7 | 73.459 | 82.540 | -0.291 |
| Scripts | 0.3 | 73.456 | 84.261 | -0.291 |
| Scripts | 0.0 | 73.269 | 83.602 | -0.307 |
| Features + Solvers Descripton + Problem Description | 0.7 | 73.224 | 80.845 | -0.311 |
| Features + Problem Description | 0.0 | 73.114 | 81.456 | -0.320 |
| Scripts + Solvers Descripton + Problem Description | 0.8 | 72.894 | 80.875 | -0.338 |
| Features + Problem Description | 0.3 | 72.678 | 82.393 | -0.356 |
| Scripts + Solvers Descripton + Problem Description | 0.2 | 71.717 | 82.042 | -0.436 |
| Features + Problem Description | 0.7 | 71.235 | 81.896 | -0.476 |
| Scripts + Solvers Descripton + Problem Description | 0.7 | 69.890 | 79.042 | -0.588 |

Table 8: Sampling-temperature sweep on the best-performing variants using `gpt-oss-120b`. "Scripts" in "Variant" column refers to `.mzn` script and instance data script. Score columns are calculated as in **??**.

## 0.6 Limited Solver Set

Throughout the course of these experiments, we manage to surpass the SBS by 1.07, reaching a closed gap of 0.08, as displayed in **??**. Still, the improvement is marginal, that's because the SBS, namely `or-tools_cp-sat-free` is widely dominant, to the point that always suggesting it wouldn't be a limiting strategy in terms of performance, as shown in **??**.

For that reason, we decided to try testing the best variants over a set of solvers with no outstandingly better models, with the aim to make the decision process for LLM as nuanced as possible, trying to understand if it actually understands the differences in problems or if it just gives as an answer what are known to be the best solvers, with a mostly arbitrary decision process.

To understand which solvers would be the best for suited for that process, we calculated the number of optimal solutions found by each of the solvers (results displayed in **??**)

Given the results, we decided to only keep the solvers with a count of optimal solutions in

| Solver | Score | Optimal Count |
|---|---|---|
| or-tools_cp-sat-free | 76.964375 | 55 |
| picatsat-free | 70.933307 | 53 |
| chuffed-free | 74.456334 | 52 |
| huub-free | 68.497987 | 48 |
| gurobi-free | 55.384518 | 38 |
| pumpkin-free | 58.543229 | 33 |
| choco-solver__cp-sat_-free | 60.808176 | 32 |
| cplex-free | 48.514529 | 30 |
| choco-solver__cp_-free | 58.404323 | 29 |
| cp_optimizer-free | 50.992682 | 26 |
| izplus-free | 58.672093 | 25 |
| jacop-free | 44.549443 | 25 |
| sicstus_prolog-free | 44.592608 | 24 |
| scip-free | 36.902766 | 20 |
| highs-free | 34.418506 | 18 |
| cbc-free | 22.615259 | 11 |
| or-tools_cp-sat_ls-free | 43.222031 | 7 |
| yuck-free | 34.715515 | 4 |
| atlantis-free | 1.500000 | 0 |

Table 9: Sampling-temperature sweep on the best-performing variants using `gpt-oss-120b`. "Scripts" in "Variant" column refers to `.mzn` script and instance data script. Score columns are calculated as in **??**.

betwee 40 and


Figure 1: Histograms displaying the performances of all the solver variants in "Closed Gap" evaluation, as calculated in **??**.


Figure 2: Heatmap displaying the performance of all the combinations of temperatures with the five best performing variants in "Closed Gap" evaluation calculated as in **??**, all tests were performed using `gpt-oss-120b`.