0.1 Methodology

In this chapterone, 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 chapter is structured into four sections, each addressing a key preliminary choice. The first section 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 final section explains the metrics adopted to evaluate how effectively each model could operate as a meta-solver.

0.2 Preliminary Large Language Models Selection

0.2.1 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 Table 2 and Table 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.

Table 1: Rate limits - Groq models[?]:
This table shows all the offered models from Groq API, in leftmost column and each relative rate limit.

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

0.2.2 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

Table 2: Rate limits - Gemini models[?]
This table shows all the offered models from Gemini 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	–

`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.

0.3 Preliminary Prompt Engineering

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.

0.3.1 General Structure

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.

0.3.2 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:

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

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 Section 0.5.

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

0.4 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.

0.5 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 $(\mathcal{I}_1, \mathcal{S}_1, \tau_1)$, $(\mathcal{I}_2, \mathcal{S}_2, \tau_2)$, $\ldots$, 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

0.5.1 Metric for Solver Score Calculation

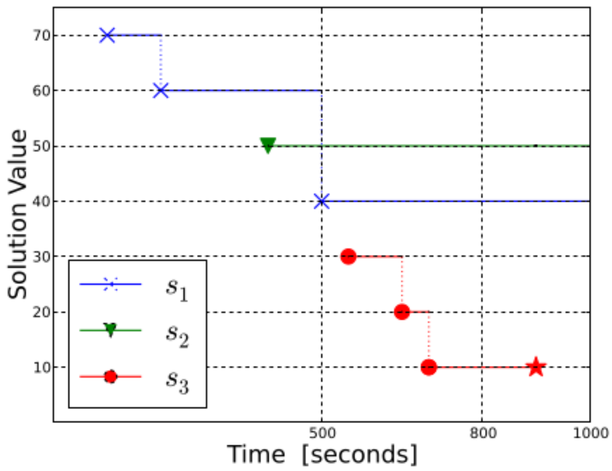

Figure 2: Solver performances example

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:

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

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), it receives the highest score.

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

0.5.2 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

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

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.