In this chapter, 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 project.

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.

0.1 Preliminary Model Selection

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 models 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**[?], 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.

- **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.

0.1.2 Model selection

Both providers expose a wide range of models with different capabilities and limitations. A preliminary filtering step was therefore required. Not all models offered by the APIs are designed for text generation, which is essential for the proposed AS. More specifically:

- `playai-tts` and `playai-tts-arabic`: text-to-speech models available only within Groq's platform and unsuitable for remote testing.

- `whisper-large-v3` and `whisper-large-v3-turbo`: speech-recognition models, irrelevant for this work.

Table 1: Rate limits - Groq models[?]

Model	RPM	RPD	TPM	TPD	ASH	ASD
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	–	–
whisper-large-v3	20	2000	–	–	7200	28800
whisper-large-v3-turbo	20	2000	–	–	7200	28800

Additional models were removed because they failed to follow instructions reliably, producing inconsistent or sometimes unreadable answers:

- `deepseek-r1-distill-llama-70b`
- `gemini-2.0-flash-lite`
- `gemma2-9b-it`
- `meta-llama/llama-prompt-guard-2-22m`
- `meta-llama/llama-prompt-guard-2-86m`
- `allam-2-7b`

After this filtering stage, eighteen models remained as a stable base for the evaluation phase.

Table 2: Rate limits - Gemini models[?]

Model	RPM	RPD	TPM	TPD
Gemini 2.5 Pro	5	100	250000	–
Gemini 2.5 Flash	10	250	250000	–
Gemini 2.5 Flash Preview	10	250	250000	–
Gemini 2.5 Flash-Lite	15	1000	250000	–
Gemini 2.5 Flash-Lite Preview	15	1000	250000	–
Gemini 2.0 Flash	15	200	1000000	–
Gemini 2.0 Flash-Lite	30	200	1000000	–

0.2 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. 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 complex prompt could influence models differently and distort the evaluation. A concise structure reduces unwanted variability.
- **Reduce token usage**: Since the testing setup depends on API limits, keeping the prompt compact minimizes token consumption.

Ensuring a standardized output format was equally important:

- **Automation**: Automated testing requires responses to follow a strict format so that solver selections can be parsed without ambiguity.
- **Context-window constraints**: Long outputs reduce the available context window for subsequent turns, making the model less usable in a multi-step system.

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**: The LLM's estimation of the single best solver.
- **Parallel-solver evaluation**: Assessing how effective the top three solvers would be if run in parallel and the best result extracted.

The metrics used for these evaluations will be detailed in Section 0.4.

After all of this considerations, the selected prompt structure is:

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

0.3 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 behaviour. 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,
  - 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* [ref]. 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 includes five corresponding data instances, yielding a total of 100 testable, diverse, and complex scenarios.

0.4 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).

But before talking about the evaluation metrics, we should spend some words on what we need to evaluate: 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 are even trickier in scenarios including optimization problems.

For those objectives two separate metrics were chosen

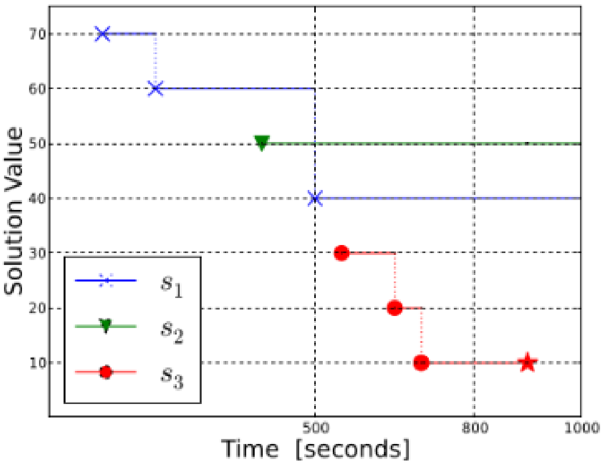

Figure 1: Solver performances example

0.4.1 Metric for Solver Score Calculation

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 Fig. 1 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 Fig. 1), 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)

- $\alpha = 0.25$
- $\beta = 0.75$

0.4.2 Closed Gap

Now that we have a way to calculate (meta-)solver score, we need a way to analyze how each of our AS does compared to the single best solver. For this objective, we have chosen to use *closed-gap*[. . . add reference] as the evaluation metric.

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

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

Where:

- m_{SBS} is the highest m_σ out of all the single solvers.
- m_{VBS} is the highest achievable total score, representing an ideally perfect solver.

The assumption $m_{\mathrm{VBS}} > m_{\mathrm{SBS}}$ is required, since otherwise a single solver would be the VBS and no algorithm selection would be needed. 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 $m_{\mathrm{SBS}} - m_{\mathrm{VBS}}$, which is nonsensical for individual solvers.

Aggiungere reference per tutta la section sulle metrics, presa in gran parte da [?]