This is the DEDICATION:
you can write whatever you want here,
or nothing at all ...

Introduction

This is the introduction.

Contents

List of Figures

List of Tables

Chapter 1

Initial Researches

This is the first chapter.

1.1 Preliminary Model Selection

1.1.1 Provider Choice

To build the proposed Agentic Solver (AS), first of all, it is needed to have the agent, so an LLM that could orchestrate the system an act as an agent. Due to limited resources and computing power, for the initial testing phase, it was decided to opt for the use of, out of the box, APIs (Application Programming Interface) offered with a free tier basis. Based "freedom of use", so less limitation on rates, the decision led to

- Gemini API (add ref) by Google DeepMind. Gemini is a family of LLMs developed by Google DeepMind, with different sizes and capabilities. The model used for the initial testing phase was Gemini 1.5 Pro, which is a powerful model that can handle complex tasks and generate high-quality text.
- Groq API by Groq (add ref). Groq is a company that provides hardware and software solutions for AI and machine learning. The Groq API allows users to access Groq's LLMs and other AI models through a simple and easy-to-use interface.

Both APIs were chosen for their ease of use, flexibility, performance and larger rate limits when compared to competitors:

1.1.2 Model selection

Out of all the provided models, a selection needs to be done. First of all not all the models provided are made for text generation:

- `playai-tts` and `playai-tts-arabic`: these models are only provided for use on groq's platform and not for remote testing

Table 1.1: 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

- `whisper-large-v3` and `whisper-large-v3-turbo`: these models are actually for speech recognition , so not useful for the purpose of this research

Other models were removed as they didn't strictly follow requests, giving inconsitent 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`

Table 1.2: 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	–

- `allam-2-7b`

After this preliminary selection we are left with a list of eighteen models as a stable base for testing.

1.2 Preliminary Prompt Engineering

To test which would be the most fitting LLM for the task of building an AS, it was needed to build a set of standard prompt to send at each model. The first thing was to chose what wold be the prompt structure, the base idea was to keep it as short and clean as possible, for two main reasons:

- **Influence the model a less as possible**: A too complex prompt structure could influence the model and possibly tweak results for certain models. So using as less words as possible is the best and easiest way to limit possible unexpected results.
- **Limit Token Usage**: The prompt needs to be limited to limit the token usage, since the testing environment is restricted to the token limits imposed from the providers.

Another important factor to consider, was to have a standardized answer, and forcing that standardization was necessary due to two main reasons:

- **Test Automation**: To make it possible to make automatic tests through programs, the answer need to have a standard form where features can be extracted seamlessly.
- **Limit Token Usage**: The token usage isn't only an issue on requests limits, another big limitation is imposed on context window dimension, so a long answer could strongly limit the model usability.

Given those considerations, the chosen output form was an array of strings like, format:

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

The choice to have the best 3 solvers as an answer was made to make two different evaluations possible:

- **Single Solver Evaluation**: A basic evaluation on what the LLM figured to be the best solver for the given problem.
- **Parallel Solver Evaluation**: An evaluation made as if all of the three selected solver were running in parallel the automatically extract the best out of the three.

The actual explanation on metrics for model evaluation on those two basis will be explained later in section 1.4

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

1.3 Problem Selection

Another clearly important decision for the testing project is the **Problem Selection**. To have consistent tests across all models, the problem selection is extremely important in order to have results that could be both significative and analyzable. In order for that to be possible, it is necessary the the list of problems has some characteristics:

- **Extensive Testing**: It needs to be proven that problems are correct, and it is needed to have actual,reliable results on solver performance on each of the given problems for consistent evaluation, preferably on last, state of the art solvers so that the system capability can be proven against the actual best competitors.
- **Diversity**: The problem list needs to be diverse, so include:
  - Combinatorial problems

  - Real-life problems
  - Puzzle-like problems

  From all three solving categories:

  - Maximization
  - Minimization
  - Satisfaction

  In order for LLM performance to be proven on every realm.

- **Complexity**: The problems need to be complex in order to make the solver choice actually significative, and in order to prove the LLM performance in a hard real life context of use.

Given those constraints the problem selection aimed to the problem list of 2025 MiniZinc Challenge[ref . . .]. The problems of the challenge have been proven to be the best on testing latest state of the art solvers. Each problem has been selected to test performance of the best solvers submitted on this same year, in order to chose the best overall solver at the moment. The problem ist is composed of a total of twenty problems:

- 1 Satisfaction problem
- 3 Maximization problems
- 16 Minimization problems

Each of those with a total of five instance relative data, resulting in a total of 100 testable, diverse and complex set of problems.

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

1.4.1 Metric for Solver Score Calculation

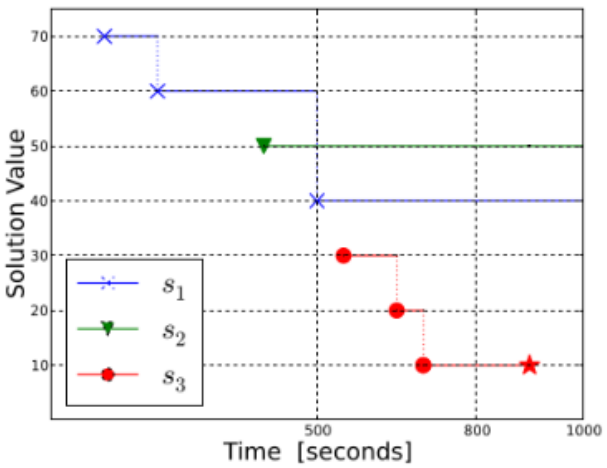

Figure 1.1: Solver performances example

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 Fig. 1.1 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)
- $\alpha = 0.25$
- $\beta = 0.75$

1.4.2 Closed Gap

Conclusions

These are the conclusions.

Appendix A

First Appendix

...

Acknowledgements

Here you can thank whoever you want.