A Collaborative Numeric Task Planning Framework based on Constraint Translations using LLMs

Anthony Favier, Ngoc La, Pulkit Verma, and **Julie A. Shah**

Massachusetts Institute of Technology

{antfav24, ntmla}@mit.edu, {pulkitv, julie_a_shah}@csail.mit.edu

Abstract

Automated planning systems require formal constraint specifications that create significant barriers for domain experts not familiar with those formal specifications, thereby limiting the practical adoption of powerful planning tools in collaborative planning settings. To overcome this challenge, we propose an LLM-based pipeline to translate human natural language constraints into formal hard-trajectory constraints. The initial user input is first refined and decomposed into more explicit natural language constraints, both preparing constraints for formal encoding and offering a chance for the human to review and correct any misinterpretation. Then, the decomposed constraints are encoded into PDDL3. By integrating this with an automated planner, a graphical interface, and PDSim, we created a closed loop where the human gets plan simulations as feedback to their natural language constraints. This innovative collaborative planning framework enables users to leverage their intuition and expertise to intuitively guide automated planning without time-consuming programming expert interventions. Through an ablation study, we demonstrate how our approach significantly improves the syntax and semantic accuracy of the translations compared to direct LLM translations. Our results demonstrate the potential of collaborative planning without technical expert interventions for higher-quality automated solving. On the other hand, our negative results seem to highlight the limitations of using PDDL3 constraints to leverage human high-level guidance as we expected, raising interesting reflections and potential discussions.

1 Introduction

Collaborative planning has shown significant potential to generate higher quality solutions and improve efficiency, particularly when incorporating soft constraints and human preferences (Wilkins and Robinson 1981; Kim, Banks, and Shah 2017). However, the accessibility of formal planning remains severely limited due to the requirement for programming knowledge or technical expert interventions. This creates a fundamental barrier that prevents domain experts, who are unfamiliar with planning formalisms, from effectively contributing their insights to the planning process. This barrier is particularly problematic for complex real-life time-constrained problem solving (e.g., disaster response scenarios). In this context, an optimal solution to such complex real-life problems (without any simplification) is usually extremely computationally challenging to obtain. Instead, the focus is put on finding the best valid solution possible, given a limited time budget.

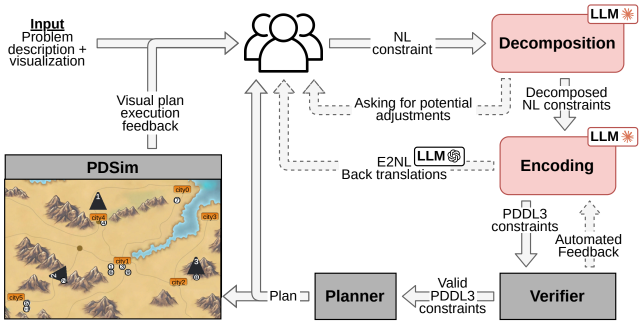


Figure 1: Overview of the collaborative planning framework

While Large Language Models (LLMs) have shown promising and continuously improving results in reasoning tasks (e.g., GPT4 improvements over GPT3.5 in mathematics and coding questions (OpenAI 2023)), they still cannot plan reliably on their own (Kambhampati et al. 2024). Nevertheless, LLMs' language processing capabilities present a unique opportunity to serve as a bridge, potentially making automated planning accessible and intuitive to anyone. Through these improvements, we can better harness human domain expertise and intuition to enhance problem-solving capabilities across various planning scenarios.

To address these challenges, we propose an innovative collaborative planning framework that includes a two-stage LLM-based pipeline to translate natural language constraints into formal planning encoding, specifically Planning Domain Definition Language 3 (PDDL3) (Gerevini and Long 2005) hard trajectory constraints. Our intuition is that reducing the search space with hard constraints can indirectly guide the search. Thus, relevant user input can act as a negative heuristic (Kibler and Morris 1981), telling the solver what not to do to avoid "intuitively bad solutions". For instance, considering a simple logistics domain, if there is no goal specified regarding a package p, then any plan moving p is intuitively inefficient. This approach also allows users to explore specific alternatives in a "Let's try this and rollback" fashion. Our approach allows domain experts to express their knowledge and preferences in natural language without requiring deep technical expertise in formal planning languages. Then, these inputs are leveraged to lead to more efficient and informed problem-solving.

Our framework integrates this translation pipeline with a state-of-the-art automated planner, a user-friendly graphical interface, and PDSim (De Pellegrin and Petrick 2024), an existing visualization tool for simulating plan executions. The overall architecture is depicted in Figure 1. The resulting collaborative planning framework creates a seamless closed-loop system where users can intuitively guide automated planning through natural language input while receiving immediate visual feedback through plan simulations. This enables users to dynamically refine solutions and iterate toward more effective outcomes in an efficient manner.

After discussing some related work in section 2, our contribution and the overall collaborative planning framework are described in sections 3 and 4. Then, section 5 reports results about the accuracy of the translating pipeline with an ablation study, demonstrating the benefits of the different elements of the pipeline. Section 6 reports mixed results about the benefits of using our approach on the solution quality. These results demonstrate the potential for significant improvements in solution quality with collaborative planning without technical expert interventions. However, our results also highlight some limitations of the current implementation with PDDL3 hard constraints. This raises interesting discussions and suggests further exploration to achieve consistent improvements, discussed in section 7.

2 Related Work

2.1 Planning with Constraints

Planning with constraints and preferences was the focus of the 5th International Planning Competition (IPC-5). PDDL3 (Gerevini and Long 2005), the language used in this competition, introduced soft (preferences) and hard constraints for both goals and trajectories. It became a standard and popular planning formalism for expressing planning problems with trajectory constraints and temporal properties. Various approaches have been explored to solve PDDL3 problems. Some solvers have been developed to natively support PDDL3, such as SGPlan (Hsu et al. 2007) or MIPS_XXL (Edelkamp, Jabbar, and Nazih 2006). Other approaches consist of compiling the PDDL3 constraints to leverage state-of-the-art planning systems and heuristics (Wright, Mattmüller, and Nebel 2018; Percassi and Gerevini 2019). Bonassi, Gerevini, and Scala (2024) propose an approach that compiles PDDL3 hard constraints to numeric planning. Paired with the ENHSP (Scala et al. 2016) numeric planner, their results seem to outperform significantly other PDDL3 planners. For this reason, we decided to use this planning approach.

We use PDDL3 to leverage the existing benchmark problems from the International Planning Competition (IPC) [1]. However, we are aware of the existence of other representations to model trajectory constraints involving mixed propositional and numeric conditions, such as Mixed Integer Linear Programming, or Signal Temporal Logic (Maler and Nickovic 2004). These methods of encoding the constraints are explored in our other ongoing projects.

[1]https://www.icaps-conference.org/competitions

2.2 Collaborative Planning

In this work, we use collaborative planning to refer to the process of collaboratively elaborating a solution to a planning problem. This is not to be confused with planning to solve a collaborative multi-agent task. Our focus is on leveraging the specific strengths of humans, LLMs, and automated planning systems to collaboratively solve planning problems, including collaborative tasks.

Kim, Banks, and Shah (2017) generate visualizations from original PDDL problems and show them to human users. The users then propose high-level strategies to solve the problem. These strategies are encoded as preferences (PDDL3 soft constraints) by technical experts. They demonstrate that using the user's high-level strategies leads to higher-quality solutions. However, this collaborative planning setup relies on manual expert translation of the natural language user input into PDDL3 soft constraints. In this work, we aim to achieve similar results while leveraging LLMs to perform these translations and bridge human users with automated planning formalism, without technical expert interventions.

2.3 Translating NL to Structured Language

Natural Language Processing (NLP), and more precisely, translating natural language to structured language such as formal queries, programming language, temporal logic, or PDDL, has been a topic of interest for decades now (Kate, Wong, and Mooney 2005; Kaufmann, Bernstein, and Fischer 2007). Transformers and Large Language Models provide impressive language processing capabilities, leading to further interest in translating natural language (NL) into structured language: programming language (Github Copilot[2] 2021; Cursor[3] 2023; Liang et al. 2023; Djuhera et al. 2025), temporal logic (Chen et al. 2023; Liu et al. 2023; Cosler et al. 2023), or PDDL (Guan et al. 2023; Gestrin, Kuhlmann, and Seipp 2024; Mahdavi et al. 2024). In several works, the translation process relies on human reviews and interventions, often requiring them to be familiar with the corresponding structured language (Cosler et al. 2023; Guan et al. 2023). In Mahdavi et al. (2024), the human reviews are 'replaced' by the assumption of having access to two functions to observe the executability and verifiability of actions, then they rely on these environment interactions to refine the generated output. Other works without external feedback (e.g. Liang et al. 2023 or Liu et al. 2023) achieve satisfactory accuracy, which convinced us to use LLMs for our translation process.

[2]https://github.com/features/copilot

[3]https://cursor.com/

3 LLM-based Translation Pipeline

The core innovation of our framework lies in a two-stage LLM-based pipeline that transforms high-level natural language constraints into precise formal planning representations. This translation process addresses the fundamental challenge of bridging the semantic gap between intuitive human expression and the rigorous requirements of automated planning systems. A key design principle of our pipeline is that users never directly interact with PDDL representations, maintaining the accessibility that motivated our approach. The model we are using for the two translation stages is Claude Sonnet 4 from Anthropic[4]. We empirically preferred the outputs of Claude over GPT to manipulate PDDL3. However, this model is interchangeable with any other LLMs. We are using zero-shot prompting, only providing the full PDDL domain and problem, some instructions, and user input.

3.1 Constraint Decomposition

The first stage of our pipeline focuses on constraint decomposition. It transforms the high-level, potentially ambiguous, natural language user input into a list of more explicit, low-level constraints. This decomposition process serves multiple critical functions in ensuring accurate translation. The system systematically rephrases user input into clearer temporal logical constraint expressions, breaks down complex constraints into manageable components, and actively resolves ambiguities that could lead to misinterpretation during the encoding phase.

The LLM is prompted with the PDDL domain and problem, an information paragraph describing that we are trying to solve the given PDDL problem and that the user will provide an NL constraint to help solve the problem, and a list of instructions indicating our desire to refine and decompose the user input constraint to match the PDDL3 formalism without explicitly using any PDDL words yet. Eventually, the user input constraint is also added to the prompt.

Through this decomposition process, constraints are refined to align more closely with Linear Temporal Logic (LTL) representations and the specific fluent structure of the planning problem domain. This refinement is essential for maintaining semantic fidelity while preparing constraints for formal encoding. A brief explanation of the decomposition choices is also generated for the user. This explanation helps understand some rephrasing that may be surprising. For instance, "Only plane1 should move" cannot directly be translated into PDDL3 due to the limited available fluents. But, once rephrased as "Any plane other than plane1 should never move", the translation is straightforward. Importantly, by sharing the natural language decomposition and explanation, this stage provides an opportunity to detect potential misinterpretations early in the process. This allows the user to provide natural language feedback to the system for clarification and correction before proceeding to formal encoding. Leveraging human interactions, this constraint decomposition stage helps to improve the semantic accuracy of our translation.

[4]https://www.anthropic.com/claude/sonnet

3.2 Formal Encoding and Verification

The second stage translates each decomposed low-level constraint into PDDL3. This encoding process leverages the clearer and more explicit constraints produced by the decomposition stage to generate syntactically correct and semantically accurate formal representations. To ensure the reliability of this translation, our pipeline incorporates an automated verifier that checks the syntax of all generated encodings. When an error is detected, an automated informed feedback is sent to the LLM for reencoding. The verifier currently performs three checks. First, it parses the generated encodings and verifies that every generated term is either a standard PDDL keyword or a fluent/object from the PDDL problem. If it's not the case, an error message reports which term was detected as incorrect. Then, the verifier checks if a temporal logic keyword is used. If not, a specific message reports this error. Eventually, a regular parsing test is performed using the unified-planning library[5]. This generates a generic error message indicating a syntax error.

The prompt follows a similar structure to the previous phase. The LLM is prompted with the PDDL domain and problem, a similar informative paragraph, and a list of instructions asking to translate the given NL constraint into PDDL3, and one of the decomposed constraints.

Additionally, our system employs an encoding-to-natural-language (E2NL) back-translation mechanism to allow users to evaluate whether the generated encodings accurately reflect each of the decomposed constraints, without requiring them to understand PDDL directly. As this back-translation process can also be erroneous, we decided to use a different language model (OpenAI o4-mini[6]) to minimize potential biases in interpreting the encodings' meaning. Indeed, a rare case we encountered, Claude Sonnet 4 generated an erroneous PDDL from an NL constraint due to a misinterpretation. When later asked to translate the PDDL back into NL under a new context, the model made the same misinterpretation and generated a sentence closely resembling the original NL constraint. Even if rare, such behavior leads to false positive encoding reviews. It seems to be mitigated by using another model for the back-translation.

4 Collaborative Planning Framework

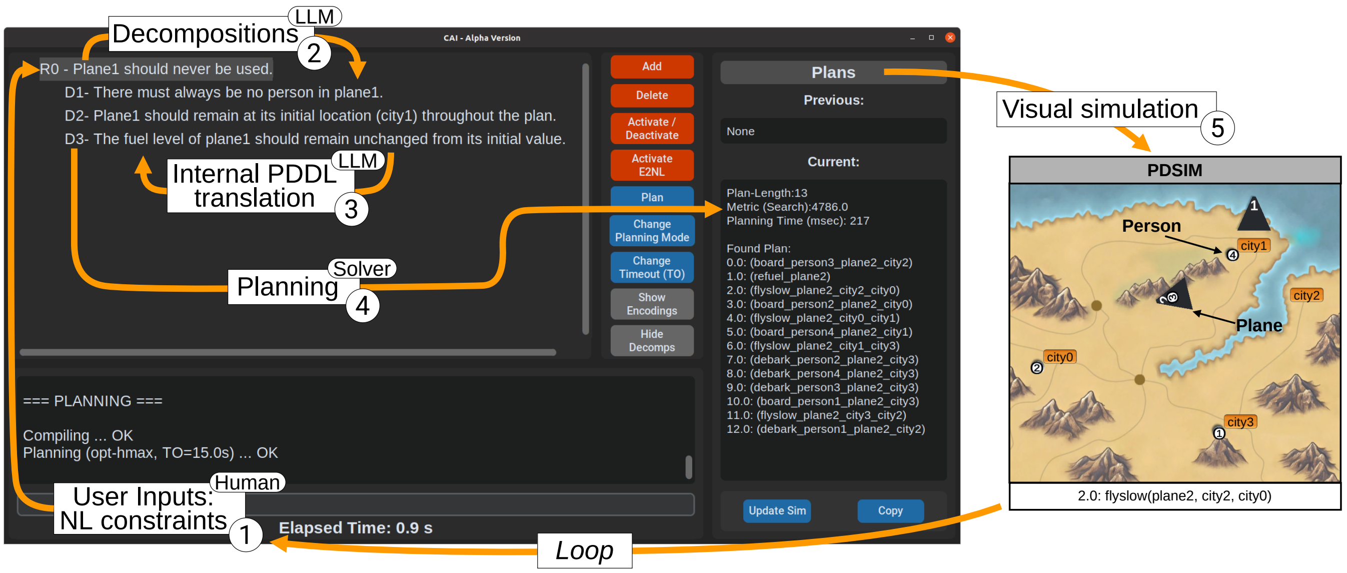
Our framework integrates the natural language translation pipeline with automated planning and visualization components to create a seamless collaborative experience. The main interface and workflow are shown in Figure 2. Below, we provide a detailed description of the workflow of our collaborative planning framework while describing its various integrated components, incorporating the LLM-powered translation pipeline presented in the previous section.

The user mostly interacts with the main interface window shown on the left side of Figure 2. It was implemented using the CustomTkinter[7] library. The workflow starts by adding the first constraint (Add button). The user provides a natural language constraint to translate using the interface (① on

[5]https://unified-planning.readthedocs.io/

[6]https://platform.openai.com/docs/models/o4-mini

[7]https://customtkinter.tomschimansky.com/



R0 - Plane1 should never be used.
D1- There must always be no person in plane1.
D2- Plane1 should remain at its initial location (city1) throughout the plan.
D3- The fuel level of plane1 should remain unchanged from its initial value.

```
=== PLANNING ===

Compiling ... OK
Planning (opt-hmax, TO=15.0s) ... OK
```

Elapsed Time: 0.9 s

```
None
```

```
Plan-Length:13
Metric (Search):4786.0
Planning Time (msec): 217

Found Plan:
0.0: (board_person3_plane2_city2)
1.0: (refuel_plane2)
2.0: (flyslow_plane2_city2_city0)
3.0: (board_person2_plane2_city0)
4.0: (flyslow_plane2_city0_city1)
5.0: (board_person4_plane2_city1)
6.0: (flyslow_plane2_city1_city3)
7.0: (debark_person2_plane2_city3)
8.0: (debark_person4_plane2_city3)
9.0: (debark_person3_plane2_city3)
10.0: (board_person1_plane2_city3)
11.0: (flyslow_plane2_city3_city2)
12.0: (debark_person1_plane2_city2)
```

Figure 2: Main interface and workflow. ① The user provides a natural language constraint. ②-③ The input is translated using our proposed interactive translation pipeline. ④ The problem is solved with the active constraints. ⑤ The generated plan is simulated in PDSim. Based on visual feedback, the human can iteratively refine the solution by adjusting the constraints.

Figure 2). The interactive LLM-powered translation pipeline starts as described in the previous section (Decomposition ② and Encoding ③). For legibility purposes, those steps are shown on the top left corner of Figure 2. However, only the final decomposition is shown on the top left, and the interactive translation happens in the bottom part. Once validated, the constraint appears in the top left corner and can be manipulated through the interface: delete or (de-)activate decomposed sub-constraints.

After setting the desired planning setting, the user can start the planning process (Plan button ④). The currently activated constraints are retrieved and their PDDL3 encodings are added to the original problem in a `:constraints` block. After, we employ NTCORE$^+$(Bonassi, Gerevini, and Scala 2024) on the updated problem and original domain to compile away the PDDL3 constraints. We obtain a compiled numeric domain and problem with conditional effects. Finally, we use the ENHSP (Scala et al. 2016) solver on the compiled files to generate a plan. Our default setting is the anytime planning mode with a fixed time budget. But the user can adjust the time budget and change to a satisficing or optimal planning mode through the interface. The planning results appear on the right side, including the generated plan, the planning time, and the plan metric/cost. If existing, these details regarding the previous plan generated are also displayed for comparison purposes.

Eventually, we use PDSim (De Pellegrin and Petrick 2024) to provide dynamic, visual simulations of plan execution (PDSim window ⑤). By default, the PDDL domain and problem must be given to the system and solved internally to visualize plan execution. Instead, we initialize the simulation with the original problem files and then automatically edit at runtime the simulation files, inserting the plan generated by our approach. This instantly updates the simulation and allows us to use the NTCORE$^+$+ ENHSP planning scheme. This visual feedback allows users to assess whether generated plans align with their intentions, identify potential issues, and iteratively refine their natural language constraints based on observed plan behaviors.

5 Translation Accuracy Results

5.1 Experiment

To measure the accuracy of our translation, we conducted an ablation study involving four progressive settings:

- ENCODING: corresponds to directly giving the human input to the encoding phase and analyzing the output. It is similar to using the LLM straightforwardly.
- VERIFIER: adds the verifier loops providing automated feedback, including parsing tests.
- DECOMP: adds the decomposition step before encoding the natural language constraints.
- HUMAN: ask the human for decomposition review and consider their feedback to re-decompose if necessary.

We considered a set of 15 predefined constraints for the IPC numeric problems ZenoTravel13 (8 constraints) and Rover10 (7 constraints). The constraints are arbitrarily more or less ambiguous. For each constraint, especially the ambiguous ones, a detailed description of the intent behind the constraint was defined. For instance, “Only use plane1” is ambiguous about what “using a plane” means. The associated intent consists of restricting all other planes from performing any action. For each setting, we run our framework to translate each constraint twice. For each PDDL3 output

Setting	Translation			Human interventions
	Parsable	Correct	Time (s) (SD)	
ENCODING	26	19	29.3 (12.3)	0
+ VERIFIER	30	20	35.8 (13.5)	0
+ DECOMP	30	20	55.0 (26.2)	0
+ HUMAN	30	27	81.9 (53.7)	12

Table 1: Ablation study reporting syntax and semantic accuracy ($N = 30$)

translation, we record its parsability, correctness, total translation time, and the number of human interventions during the translation process. Parsability was measured by updating the original PDDL problem with the encoded constraint and attempting to parse the new problem using the unified-planning library. Since there exist several valid translations for each NL constraint, the correctness was evaluated manually in a binary way, as either properly matching the associated human intent or not. The translation time starts from the human typing the input to the final PDDL3 constraint. The number of human interventions corresponds to the number of times the human had to provide feedback on the decomposition to correct a misinterpretation or an incomplete decomposition.

The human interventions with the HUMAN setting were as simple as possible. They either clarify a clear misinterpretation or point out an incomplete decomposition. For instance, the constraint "Rock analysis from waypoint1 should be performed first" was misinterpreted as "The first rock analysis performed should be from waypoint1", while the human intent is, in fact, "Among all subgoals, obtaining rock analysis from waypoint1 should be done first". The human intervention here was: "I am referring to the whole goal, rock analysis from waypoint1 should be the first goal accomplished". As another example, the decomposition of the constraint "Only use plane1" mentions preventing other planes from flying and boarding/debarking persons, but not preventing them from being refueled. The human intervention was: "Also prevent the other planes from being refueled".

This experiment has been conducted by the same person, part of the authors. Moving forward, we plan to conduct a user study, which will strengthen our results. In the user study, users' typing speed is a variable to consider with our framework. However, in this paper, its effects haven't been studied as the experimenter's typing speed was more or less constant.

We empirically identified that several intuitive constraints rely on tracking action occurrences. Thus, the two domains were augmented with additional fluents counting action occurrences.

The major LLM model used is Anthropic Claude Sonnet 4 20250514 with thinking enabled. We also use GPT o4-mini for the E2NL back-translation process, described in Sec. 3.

5.2 Results

Results are reported in Table 1. The row of ENCODING setting shows that straightforwardly using the LLM sometimes fails to produce parsable PDDL (only 26 out of 30 are parsable), which makes the output unusable. We also obtain only 19 correct outputs out of the 30 tests. A non-parsable output is considered incorrect. But the poor correctness is mostly due to some missing aspects of the human intent or misinterpretations of the natural language constraints.

In VERIFIER setting, the LLM leverages the automated verifier to always produce parsable PDDL outputs. Over the 30 runs, 9 automated feedback were generated affecting 5 different constraints. Fixing the syntax of the outputs led to additional correct encodings, which only increased the number of correct outputs by 1.

Results in DECOMP setting indicate that integrating the decomposition step slows the translation and doesn't directly improve the correctness. This step can be seen as a sort of chain of thought, which can be beneficial for small models. Although here we are using a self-reflection-enabled model, which is already conducting a similar internal chain of thoughts. We believe this is why no improvement is observed in our case. Nevertheless, the natural language decomposition can be leveraged through human user review and feedback.

In HUMAN setting, the human reviews and feedback permitted by the decomposition step take additional time, but increase the number of correct outputs to 27. The 12 human interventions over the 30 runs permitted to have decompositions matching the human intent for every constraint. The 3 failures are due to small mistakes in the generated PDDL (encoding phase), leading to parsable but incorrect encodings. Particularly, the LLM usually inverts the two arguments of the `sometime-before <after_predicate> <before_predicate>` constraint, leading to an erroneous encoding. Conducting a Fisher's exact test between DECOMP and HUMAN on correctness, we obtain a one-tailed p-value $= 0.029 < 0.05$. This indicates that there is a significant improvement in correctness when leveraging human feedback.

The final average translation time of approximately $82s$ (SD=54) is lower than the average technical expert translation time of $180s$ (SD=78) given in Kim, Banks, and Shah 2017. The comparison isn't fair because we are translating similar but different constraints. Still, this suggests that our framework seems to translate NL inputs into PDDL3 faster than technical experts.

E2NL: We only provide preliminary results for the use of the back-translation (E2NL). When used on the encoded decomposed constraint, it sometimes permits the detection of erroneous encoding, especially after several loops with the verifier. If used without the decomposition, thus on the whole encoding only using the verifier, the back-translation can be too summarized and not properly reflect the generated PDDL. Also, human feedback loops on the whole encoding can lead to omitted parts, due to the LLM being focused on the specific part mentioned in human feedback. Using the decomposition helps to avoid such omission and provides more robustness.

6 Solution Quality Results

In this section, we examine the impact of our approach on the quality of the plan produced within a limited time budget. More precisely, we analyse how our approach induces translation and compilation delays but can lead to more efficient solving and better solutions. We provide some details on the experiment before presenting and discussing our mixed results.

6.1 Experiment

We considered numeric problems from the IPC. Particularly, the ZenoTravel and Rover domains. We focused our results on numeric ZenoTravel13 and Rover10, which are hard enough problems to require a long time (more than 30min) to be solved optimally on our setup using the ENHSP solver. All results were computed on a laptop equipped with an Intel Core i9-10885H 2.40-5.30GHz and 64GB RAM.

Given a limited time budget from $50s$ to $600s$, we compared our approach with two baselines. The first one is to solve the original problem directly using the ENHSP solver (labeled `original`). We solve the original problem 10 times for each time budget. Comparing our approach with this baseline could demonstrate whether it is worth spending time involving the human in the loop to plan more efficiently.

The second baseline consists of solving the problem with random constraints (labeled `random`). Comparing with this baseline will indicate if our positive results are simply due to planning with arbitrary constraints (reducing the problem's complexity even if possibly degrading the best reachable solution) or if it is by leveraging the human in the loop. The random constraints were generated by selecting random expressions from the initial state and enforcing these expressions to remain true. We obtain "simple" hard trajectory constraints. In the same manner, we also generate constraints with two or three random expressions combined with 'and' or 'or' logic. We obtain 5 sets of constraints, respectively: simple, and2, and3, or2, or3, each containing 6 constraints for a total of 30 random constraints. The problem is solved once with each constraint. The sets of constraints were generated once for each problem and are the same for every time budget. Note that in our planning scheme, using constraints implies first compiling the PDDL3 numeric problem into a PDDL2.1 problem using NTCORE$^+$. This step is included in the provided time budget and thus reduces the effective planning time.

For our approach (labeled `human`), a human provided 5 constraints for each problem, and each constraint has been translated using our pipeline. These human constraints are intuitive, complementary, and their purpose is to avoid exploring "bad" solutions. Considering every possible combination of these initial 5 constraints (using 'and' logic), we generate a set of 31 human constraints for each problem. We then solved the problem once using each constraint. For ZenoTravel13, we used the following human constraints:

- "Only use plane1"
- "Person7 should never move"
- "Planes should only fly slowly"
- "Plane1 should never fly to the same city more than 3 times"
- "Person1 and person3 should travel together".

For Rover10, we used:

- "Rover2 should never be used"
- "Rover0 should handle soil and rock data from waypoint4"
- "No rover should ever be in waypoint2 or waypoint5"
- "Rover1 should take all images"
- "Waypoint6 should always have the same rock sample"

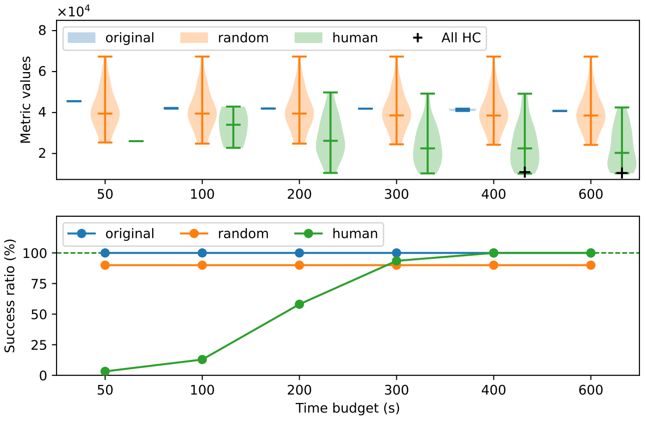


Figure 3: Solution quality results for ZenoTravel13. Shows the plan metric value (lower the better) and coverage for several time budgets and for our approach (`human`) and two baselines (`original` and `random`). The black crosses show the metric obtained when using all human constraints at once (All HC). Note that the x-axis is not linear.

Similarly to `random`, our approach induces some delay by compiling the constraints. Moreover, additional significant delays are induced due to the translation step. All these delays are included in the time budget, again reducing the effective planning time but allowing a fair comparison with `original`.

For each time budget and system, we report the metric value of the plan obtained (lower values are better) and the coverage. For ZenoTravel, the metric corresponds to the total fuel consumed, and for Rover corresponds to the total energy used. Similar to the previous section, we augmented the domains with additional fluents to track action occurrences.

6.2 Positive Results

Our results for ZenoTravel13 are reported in Figure 3. We first discuss positive results from this experiment, demonstrating the potential of our approach to leverage human in the loop for improved problem solving, with a limited budget.

Solving directly the original problem (`original`) gives very consistent results with a coverage of 100%. Regarding solution quality, the fuel consumed slowly improves with longer time budgets, going from $45,537$ for $50s$ to $40,769$ for $600s$. Note that `original` can consistently find the same initial solution with a time budget of only $10s$.

For the `random` baseline, the average compilation time of a constraint is $3.14s$ (SD=0.50), which is negligible for the time budgets considered here. As expected, `random` generates a wide diversity of results. The points higher than `original` demonstrate that the random constraints can complexify the problem, leading to worse solutions for the same time budget. Additionally, the lower and constant `random` coverage of 90% is due to 3 random constraints making the problem unsolvable. However, the metrics lower than `original` demonstrate that using constraints can lead to better solutions for the same time budget. For instance, with a time budget of $50s$, the best metric obtained with random constraints is $25,381$, which is almost half of the metric obtained with `original`. This supports our approach of using "relevant" constraints to facilitate the search and find better solutions. Mann-Whitney U tests indicate that the metrics obtained with `random` are not significantly better than `original`, except for a time budget of $50s$. We obtain the following respective one-tailed p-values for every time budget: $0.013, 0.27, 0.31, 0.20, 0.22, 0.31$.

The compilation time for `human` is $7.83s$ (SD=2.87). This is slightly longer than for the random constraints and is expected due to the more complex nature of the human constraints. But this is still negligible for the time budgets considered. However, the translation time is $167.1s$ (SD=69.6), ranging from $42.2s$ for the "easiest" constraint to $323.8s$ for all constraints at once. This significant translation time affects the coverage. Any time budget lower than $330s$ can't lead to a 100% coverage. Also, note that the metrics shown only reflect the corresponding coverage, e.g., `human` at $50s$ only has one data point. When considering time budgets sufficient for full coverage ($>=400s$), we can observe that using our approach can lead to better plans. The best metric is now $10,356$. This indicates that the additional delay induced by the human in the loop and constraint compilation can be worthwhile and can improve the problem-solving process. Mann-Whitney U tests indicate that the metrics obtained with `human` are always significantly better than `original`, except for the time budget $100s$. The respective one-tailed p-values obtained for every time budget are: $0.002, 0.059, < 0.001, < 0.001, < 0.001, < 0.001$.

6.3 Negative Results

Despite showing the potential of our collaborative planning scheme, our experiments also gave some negative results. Before going further, we remind that the human constraints considered here are well thought out and supposed to help the search by adding hard trajectory constraints to avoid "bad" solutions. Hence, intuitively, the more constraints are used, the more limited and easier the search becomes. However, our observations demonstrate that the human constraints are not as beneficial as expected.

First, we can see on Figure 3 that some combinations of human constraints lead to worse solutions than `original` (time budgets $>= 200s$). One could think this is because each constraint has a different effective planning time due to the different translation and compilation delays. However, in practice, one of the human constraints with an effective planning time of $68.9s$ produced a plan with a metric of $49,822$, while another constraint with a shorter effective planning time of $49.7s$ led to a better metric of $26,149$. This indicates that human constraints don't have the systematic beneficial

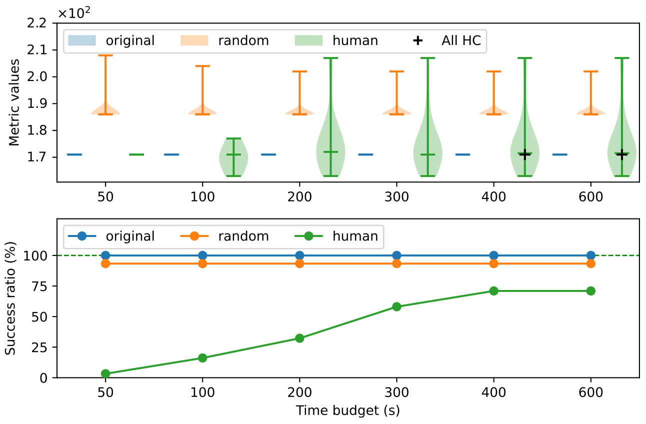


Figure 4: Solution quality results for Rover10. Shows the plan metric value (lower the better) and coverage for several time budgets and for our approach (`human`) and two baselines (`original` and `random`). The black crosses show the metric obtained when using all human constraints at once (All HC). Note that the x-axis is not linear.

effect that we expected.

Our results using the numeric Rover10 problem are reported in Figure 4. They also show that using specific constraints can lead to better solutions. However, they also provide more negative results as `random` never does better than `original`, and `human` doesn't lead to much better solutions than `original`. Additionally, human constraint now can lead to even worse solutions than `random`. Mann-Whitney U tests confirm that for every time budget, `human` doesn't lead to significantly better metrics, in contrast to ZenoTravel13. Moreover, we marked with black crosses the metric obtained when using all human constraints at once. We observe that even with big time budgets, some constraints lead to better solutions than when using all constraints at once. This means that there is an optimal subset of constraints to use to get the best solution, and just using as many constraints as possible doesn't necessarily improve problem-solving.

Unexpected results about the coverage are worth mentioning. For both problems, `random` sometimes makes the problem unsolvable, leading to the coverage upper bound observable on Figures 3 and 4. However, we can observe a similar upper bound for `human` in Figure 4 for $400s$ and $600s$. The human constraints can't lead to an unsolvable problem. Instead, here the solver 'timeouts' and no solution is found within the given time budget. We were not able to identify a pattern explaining why some constraint combinations lead to no solution.

For reference, we manually generated a plan for Rover10 satisfying all human constraints used and reaching a metric of 139. This is better than any obtained results and shows that the constraints used can lead to significantly better solutions. We also used the ZenoTravel7 problem with human constraints very close to the ones used for ZenoTravel13. We were able to confirm that the constraints are all satisfied by the optimal solution and thus shouldn't obstruct the search.

Nevertheless, we observe the same kind of mixed results.

Another interesting result is regarding the use of the PDDL3 compilation planning scheme. For Rover, the average compilation time for `random` and `human` was $34.3s$ (SD=3.3) and $31.6s$ (SD=2.5) respectively. This is ten times longer than `random` for ZenoTravel13. This shows that the compilation time isn't always negligible in this planning approach. Moreover, we also considered using the Woodworking and Parking domains, but the compilation step, even for simple problems and constraints, was taking more than an hour and had to be stopped. The reason behind this long duration is yet to be determined.

7 Discussion and Conclusion

Our experiments show that despite intuitive and legitimate human inputs, the effects on problem solving are not always positive and can even be negative. Our main intuition regarding this matter is that our choice of compiled PDDL3 planning approach doesn't suit our goals. Despite technically reducing the search space and satisfying the given constraints, we believe the constraints might be conflicting with the planner's heuristic and thus misguiding the search. As a result, even hard trajectory constraints avoiding "bad" solutions don't necessarily improve performance. These findings indicate we can't leverage human input as we expected using this PDDL3 constraint planning scheme.

Nonetheless, our results demonstrate that our framework can effectively incorporate human natural inputs as planning constraints with satisfactory accuracy without technical expert interventions. Our results also suggest that our framework is faster at translating than technical experts. In that sense, by conducting additional experiments, we should be able to demonstrate significant time gains of our integrated solution over a setup where users have to manually interact with LLM Chatbots and update problem files.

Using PDDL3 constraints means that human inputs must be translatable to an LTL-like (linear temporal logic) representation. This state-based representation can be limited, especially when the problem description lacks sufficient fluents to encode the constraint or when human constraints are directly related to actions and not states. We proposed to augment the problem with additional fluents to keep track of action execution, but this is still limited.

Common misconceptions have been identified when using LTL (Greenman et al. 2022). Such misconceptions probably also apply to PDDL3. This implies that even technical expert translations can be erroneous. Also, explicitly accounting for the identified common misconceptions could improve the translation accuracy.

Nevertheless, our results using our collaborative planning framework still demonstrate the potential to bridge human intuition and formal automated planning through accessible interfaces. Compared to solving directly the original problem, we were able to achieve significant improvements in solution quality within the same time budget. We also demonstrated how our pipeline leads to significantly better translations, both syntactically and semantically, compared to direct LLM translations.

Our overall approach relies on human input, which creates a trade-off: updated problems can become harder to solve or even unsolvable, but this allows humans to gain insights on specific suboptimal alternatives while avoiding "common-sense" inefficient solutions.

Our findings suggest that the PDDL3 constraint planning seems to be limited for our goals, but we are actively working on exploring other ways to leverage high-level human inputs for collaborative planning.

Additionally, in this work, we consider only the time to translate and use constraints, but not how to come up with the constraints. In this regard, it would be interesting to study how "good" humans are at providing helpful input to the framework and what feedback the system might offer to assist them.

Acknowledgments

This work was supported in part by the ONR under grant N000142312883.

References

Bonassi, L.; Gerevini, A. E.; and Scala, E. 2024. Dealing with Numeric and Metric Time Constraints in PDDL3 via Compilation to Numeric Planning. *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI).*

Chen, Y.; Gandhi, R.; Zhang, Y.; and Fan, C. 2023. NL2TL: Transforming Natural Languages to Temporal Logics using Large Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP).*

Cosler, M.; Hahn, C.; Mendoza, D.; Schmitt, F.; and Trippel, C. 2023. nl2spec: Interactively Translating Unstructured Natural Language to Temporal Logics with Large Language Models. In *Proceedings of the 35th International Conference on Computer Aided Verification (CAV).*

De Pellegrin, E.; and Petrick, R. P. 2024. Planning Domain Simulation: An Interactive System for Plan Visualisation. In *Proceedings of the 34th International Conference on Automated Planning and Scheduling (ICAPS).*

Djuhera, A.; Seffo, A.; Asai, M.; and Boche, H. 2025. "Don't Do That!": Guiding Embodied Systems through Large Language Model-based Constraint Generation. *arXiv preprint arXiv:2506.04500.*

Edelkamp, S.; Jabbar, S.; and Nazih, M. 2006. Large-scale optimal PDDL3 planning with MIPS-XXL. *5th International Planning Competition Booklet (IPC-2006).*

Gerevini, A.; and Long, D. 2005. Plan Constraints and Preferences in PDDL 3: The Language of the Fifth International Planning Competition. Technical Report RT-2005-08-47, Dipartimento di Elettronica per l' Automazione, Universitá di Brescia, Italy.

Gestrin, E.; Kuhlmann, M.; and Seipp, J. 2024. NL2Plan: Robust LLM-Driven Planning from Minimal Text Descriptions. In *ICAPS 2024 Workshop on Human-Aware and Explainable Planning (HAXP).*

Greenman, B.; Saarinen, S.; Nelson, T.; and Krishnamurthi, S. 2022. Little Tricky Logic: Misconceptions in the Understanding of LTL. *The Art, Science, and Engineering of Programming*.

Guan, L.; Valmeekam, K.; Sreedharan, S.; and Kambhampati, S. 2023. Leveraging Pre-trained Large Language Models to Construct and Utilize World Models for Model-based Task Planning. In *Proceedings of the 37th Annual Conference on Neural Information Processing Systems (NeurIPS)*.

Hsu, C.-W.; Wah, B. W.; Huang, R.; and Chen, Y. 2007. Constraint Partitioning for Solving Planning Problems with Trajectory Constraints and Goal Preferences. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence (IJCAI)*.

Kambhampati, S.; Valmeekam, K.; Guan, L.; Verma, M.; Stechly, K.; Bhambri, S.; Saldyt, L. P.; and Murthy, A. B. 2024. Position: LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*.

Kate, R. J.; Wong, Y. W.; and Mooney, R. J. 2005. Learning to Transform Natural to Formal Languages. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*.

Kaufmann, E.; Bernstein, A.; and Fischer, L. 2007. NLP-Reduce: A "naive" but Domain-independent Natural Language Interface for Querying Ontologies. In *Proceedings of the 4th European Semantic Web Conference (ESWC)*.

Kibler, D. F.; and Morris, P. 1981. Don't Be Stupid. In *IJCAI*.

Kim, J.; Banks, C.; and Shah, J. 2017. Collaborative Planning with Encoding of Users' High-Level Strategies. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*.

Liang, J.; Huang, W.; Xia, F.; Xu, P.; Hausman, K.; Ichter, B.; Florence, P.; and Zeng, A. 2023. Code as Policies: Language Model Programs for Embodied Control. In *Proceedings of the 2023 IEEE International Conference on Robotics and Automation (ICRA)*.

Liu, J. X.; Yang, Z.; Idrees, I.; Liang, S.; Schornstein, B.; Tellex, S.; and Shah, A. 2023. Grounding Complex Natural Language Commands for Temporal Tasks in Unseen Environments. In *Proceedings of the 7th Annual Conference on Robot Learning (CoRL)*.

Mahdavi, S.; Aoki, R.; Tang, K.; and Cao, Y. 2024. Leveraging Environment Interaction for Automated PDDL Translation and Planning with Large Language Models. In *Proceedings of the 38th Annual Conference on Neural Information Processing Systems (NeurIPS)*.

Maler, O.; and Nickovic, D. 2004. Monitoring Temporal Properties of Continuous Signals. In *Proceedings of the International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*.

OpenAI. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Percassi, F.; and Gerevini, A. E. 2019. On Compiling Away PDDL3 Soft Trajectory Constraints without Using Automata. In *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS)*.

Scala, E.; Haslum, P.; Thiebaux, S.; and Ramirez, M. 2016. Interval-Based Relaxation for General Numeric Planning. In *Proceedings of the 32nd European Conference on Artificial Intelligence (ECAI)*.

Wilkins, D. E.; and Robinson, A. E. 1981. An Interactive Planning System. Technical Report 245, SRI International.

Wright, B.; Mattmüller, R.; and Nebel, B. 2018. Compiling Away Soft Trajectory Constraints in Planning. In *Proceedings of the 16th International Conference on Principles Of Knowledge Representation And Reasoning (KR)*.

A Appendix: Prompts

This appendix provides the prompts used for the various LLM queries made by our framework.

A.1 Decomposition

The decomposition phase uses the following prompt to start the decomposition. The PDDL domain and problem are inserted in {`domain`} and {`problem`}. The user constraint input is inserted in {`constraint`}.

Decomposition Prompt

```
<documents>
    <pddl_domain>
    {domain}
    </pddl_domain>

    <pddl_problem>
    {problem}
    </pddl_problem>
</documents>

<information>
The user will give as input a constraint in natural language. This constraint must be
    used as a trajectory hard constraint for the solution plan of the given PDDL
    problem.
</information>

<instructions>
- Refine the user constraint to make it applicable to the given PDDL problem.
- You can rephrase and decompose the initial constraint into several other constraints.
- These constraints should be complementary, and they will be later combined with AND
    operators.
- Constraints must be state-based and follow a Linear Temporal Logic. So constraints
    can't directly refer to actions.
- Avoid explicit PDDL language in your answer, the user can't understand PDDL.
- The set of all refined constraints must capture the meaning of the initial user
    constraint.
- Format your answer as a clear and concise numbered list between the tags
    <constraints> and </constraints>. There should be no subitems, only the list of
    refined constraints.
- After, add 1 or 2 sentences to explain your choice of decomposition between the tags
    <explanation> and </explanation>.
- Here is an example of how to format your answer:
    <constraints>
        1. [natural_language_constraint_1]
        2. [natural_language_constraint_2]
        ...
    </constraints>
    <explanation>
        [concise_explanation]
    </explanation>
</instructions>

<user_input>
{constraint}
</user_input>
```

If not satisfied with the generated decomposition, the LLM is prompted as follows to re-decompose the constraint, considering the user feedback {`feedback`}. This shares the same context window as the initial decomposition.

Re-decompose Prompt

```
[previous context]

Decompose again the constraint while considering the following: {feedback}
```

A.2 Encoding

The encoding phase uses the following prompt to encode a given constraint into PDDL3.0. The PDDL domain and problem are inserted in {domain} and {problem}. The constraint to encode is inserted in {constraint}. It corresponds to each of the sub-constraints generated by the decomposition phase.

Encoding Prompt

```
<documents>
    <pddl_domain>
    {domain}
    </pddl_domain>

    <pddl_problem>
    {problem}
    </pddl_problem>
</documents>

<information>
The user will give as input a natural language constraint that must be translated into
    PDDL3.0. The translation will be used by a PDDL planner.
</information>

<instructions>
- Translate the input constraint into correct PDDL3.0.
- The resulting PDDL3.0 constraint must capture the same meaning as the initial input
    constraint.
- Remember that PDDL3.0 constraints are state-based. They can only refer to existing
    predicates and fluents, thus, not to actions.
- Be sure to include temporal logic like operators in your translation.
- Format your answer such that there is no preamble and such that the PDDL translation
    is between the tags <pddl> and </pddl>.
</instructions>

<user_input>
{constraint}
</user_input>
```

If the automated verifier detects an error in the generated PDDL, one of the three following prompts is used with the same context window. After “manually” parsing the generated PDDL, if a word is neither a standard PDDL keyword nor the name of an object or fluent from the problem, then the first prompt is used, replacing {x} with the detected word. If no PDDL3 temporal logic keyword (e.g., always, sometime) is detected, then the second prompt is used. Eventually, if the final parsing test using the unified-planning library fails, the third prompt is used. The error messages from that parsing process are rarely informative. Therefore, we use a fixed prompt.

Re-Encoding based on Verifier automated feedback Prompt

```
[previous context]

{x} is not a supported PDDL keyword or part of the problem description. Try again to
    translate correctly.
---
There is no temporal logic keyword. This is mandatory for a correct PDDL3.0
    constraint. Try again.
---
```

```
There is a syntax error. Try again carefully to translate correctly.
```

A.3 Back-Translation (E2NL)

The back-translation (E2NL) uses the following prompt:

Back-translation (E2NL) Prompt

```
<documents>
    <pddl_domain>
    {domain}
    </pddl_domain>

    <pddl_problem>
    {problem}
    </pddl_problem>
</documents>

<information>
The user will give as input PDDL3.0 constraints that must be translated in natural
    language.
</information>

<instructions>
- Translate the user input into natural language.
- Your translation should closely match the PDDL3.0 input, without additional
    deductions or reasoning.
- Your answer should be concise and not exceed 3 sentences.
- Your translation should not contain any explicit PDDL element, the user can't
    understand PDDL.
- Format your answer such that your translation is between the tags <E2NL> and </E2NL>.
</instructions>

<user_input>
{constraint}
</user_input>
```

The re-encoding based on human feedback after E2NL is done with a prompt only including the user input, and using the same context window.

B Appendix: Statistical Tests

This appendix provides details about the statistical tests computed in our results.

B.1 Translation Accuracy Results (Section: 5.2)

This section provides details about the statistical tests made when analyzing the correctness of translations during the ablation study presented in section 5.2.

NORMALITY TESTS: To test the normality of our data for the translation correctness analysis, we used the D'Agostino and Pearson's test. The results for each setting of the ablation study are reported below in Table 2:

Setting	p-value
Encoding	**8.84e-05**
+ Verifier	**6.84e-05**
+ Decomp	**6.84e-05**
+ Human	**5.18e-08**

Table 2: Correctness analysis: Normality tests. (Bold values indicate normality rejection)

The null hypothesis of normal distribution has been rejected for each setting. This indicates with high confidence that our data doesn't follow a normal distribution.

HOMOGENEITY TESTS: We also check the homogeneity of our data. Considering its non-normality, we use Levene tests for each pair of settings. The results are reported below in Table 3:

Pair of settings	p-value
Verifier-Encoding	0.791
Decomp-Encoding	0.791
Decomp-Verifier	1.000
Human-Encoding	**0.014**
Human-Verifier	**0.028**
Human-Decomp	**0.028**

Table 3: Correctness analysis: Homogeneity tests. (Bold values indicate homogeneity rejection)

The results indicate that the three pairs of data group including the Human setting have rejected the null hypothesis of similar variance. This indicates with high confidence that Human has a significantly different variance than the others.

COMPARISON TESTS: Considering the non-normality and inconsistent homogeneity of our data, we use a nonparametric statistical test. We decided to use Fisher's Exact Test because it does not make any assumptions about the underlying distribution of the data. We computed for each pair of settings the two-sided and both one-tailed (less and greater) p-values. When lower than 0.05, these p-values indicate respectively if the two settings have significantly different values, and if the left setting of the pair has significantly lower (or greater) values than the right setting of the pair. The results for each pair of settings are reported below in Table 4

Pair of settings	p-value		
	two-sided	less	greater
Verifier-Encoding	1.000	0.706	0.500
Decomp-Encoding	1.000	0.706	0.500
Decomp-Verifier	1.000	0.608	0.608
Human-Encoding	**0.030**	0.997	**0.015**
Human-Verifier	0.057	0.995	**0.029**
Human-Decomp	0.057	0.995	**0.029**

Table 4: Correctness analysis: Comparison tests. (Bold values indicate statistical significance)

The results indicate that the Human setting has significantly higher translation correctness than all other settings.

B.2 Solution Quality - Positive Results (sec: 6.2)

This section provides details about the statistical tests made when analyzing the quality of obtained solutions when the experiment presented in section 6.2 with the ZenoTravel13 problem.

NORMALITY TESTS: To test the normality of our data for the translation correctness analysis, we used the D'Agostino and Pearson's test. The results for each group (`original`, `random`, `human`) and each time budget are reported below in Table 5:

setting	time budget (s)					
	50	100	200	300	400	600
original	**nan**	**6.3e-7**	**nan**	**nan**	**0.049**	**nan**
random	0.308	0.307	0.288	0.197	0.133	0.111
human	**nan**	**nan**	0.741	0.298	0.273	0.133

Table 5: Solution quality analysis (ZenoTravel13): Normality tests. (Bold values indicate normality rejection or uncertainty)

The normality test results are inconsistent. The non-normality of data is ensured for `original` with a time budget of 100s or 400s. The 'nan' values indicate that the test couldn't be properly conducted.

COMPARISON TESTS: Based on the inconsistent normality of our data, we decided to use the nonparametric Mann-Whitney U rank test. The results for each group and each time budget are reported below in Table 6

setting	time budget (s)					
	50	100	200	300	400	600
random < original	**0.013**	0.273	0.308	0.199	0.308	0.308
human < original	**2.2e-3**	0.059	**3.2e-4**	**2.7e-5**	**7.8e-5**	**1.9e-5**
human < random	0.069	0.092	**1.4e-4**	**3.1e-7**	**6.2e-7**	**3.2e-7**

Table 6: Solution quality analysis (ZenoTravel13): One-tailed less p-values with Mann-Whitney U rank tests. Bold values indicate that for the corresponding time budget, the left setting generated significantly better plans than the right setting.

The results indicates that with enough time budget ($\geq 200s$) `human` generates significantly better solutions than other settings.

B.3 Solution Quality - Negative Results (sec: 6.3)

This section provides details about the statistical tests made when analyzing the quality of obtained solutions when the experiment presented in section 6.3 with the Rover10 problem.

NORMALITY TESTS: To test the normality of our data for the translation correctness analysis, we used the D'Agostino and Pearson's test. The results for each group (`original`, `random`, `human`) and each time budget are reported below in Table 7:

setting	time budget (s)					
	50	100	200	300	400	600
original	**nan**	**nan**	**nan**	**nan**	**nan**	**nan**
random	**1.6e-15**	**1.6e-15**	**1.6e-15**	**1.6e-15**	**1.6e-15**	**1.6e-15**
human	**nan**	**1.6e-4**	**1.6e-4**	**1.6e-4**	**1.6e-4**	**1.6e-4**

Table 7: Solution quality analysis (Rover10): Normality tests. (Bold values indicate normality rejection or uncertainty)

The normality test results indicate that our data is mostly not following a normal distribution. The 'nan' values indicate that the test couldn't be properly conducted.

COMPARISON TESTS: Based on the non-normality of our data, we decided to use the nonparametric Mann-Whitney U rank test. The results for each group and each time budget are reported below in Table 8

setting	time budget (s)					
	50	100	200	300	400	600
random < original	0.999	0.999	0.999	0.999	0.999	0.999
human < original	1.0	0.215	0.915	0.862	0.941	0.941
human < random	**1.2e-4**	**1.1e-7**	**1.9e-6**	**3.5e-7**	**8.4e-7**	**8.4e-7**

Table 8: Solution quality analysis (Rover10): One-tailed less p-values with Mann-Whitney U rank tests. Bold values indicate that for the corresponding time budget, the left setting generated significantly better plans than the right setting.

The results indicate that `human` only generated significantly better plans than `random`. But the former didn't perform better than `original`. Indeed, `random` performed significantly worse than `original` in this setup.