Automated Repair of Totally-Ordered Hierarchical Task Network Domains via Context-Free Grammars with Large Language Model Support

Daniel Lutalo, Pascal Bercher

School of Computing, The Australian National University, Canberra, Australia
daniel.lutalo@anu.edu.au, pascal.bercher@anu.edu.au

Abstract

Repairing flawed domain models remains a critical challenge in AI planning, with few effective techniques available. We propose a novel approach for repairing totally ordered hierarchical task network (TO-HTN) models with missing actions, guided by a plan that must be valid for the repaired model. This problem has only one previously documented approach, which relies on complex re-encoding that's solved via TO-HTN planning. In contrast, our approach translates the repair task into a context-free grammar repair problem and leverages a large language model (LLM) to identify and insert relevant actions directly, simplifying the repair process. We evaluate our approach on established benchmarks and demonstrate substantially improved results over the prior approach, achieving nearly three times the number of instances solved, and nearly solving all instances of domains in which the previous approach solved zero. Importantly, we mask all natural language hints, such as action names, forcing the LLM to simulate reasoning and planning, and mitigating the risk of data leakage from its training corpus.

Introduction

Automated planning now solves many complex problems, yet modelling planning domains remains difficult and error-prone (McCluskey, Richardson, and Simpson 2002). Increased modelling support can reduce the time and effort required to model planning problems and lower barriers for practitioners beyond academia. We study a modelling support setting where a plausible plan is known, but some required model interactions have not yet been anticipated.

While much progress has been made in plan generation, recent work by Langley (2025) underscores that the acquisition of hierarchical task knowledge remains a major bottleneck. Langley reviews the benefits of hierarchical task decomposition and identifies key challenges for learning HTNs automatically, such as discovering reusable hierarchical structures, unifying methods, and inferring their applicability. Despite advances in machine learning and domain-independent planning, manually engineering hierarchical knowledge continues to be tedious, error-prone, and a barrier to scaling planning solutions. This motivates the development of more automated and robust approaches for repairing or acquiring hierarchical domain models.

Hierarchical planning (Erol, Hendler, and Nau 1996; Bercher, Alford, and Höller 2019) is a key extension of classical planning and is strictly more expressive (Höller et al. 2014, 2016; Alford et al. 2016; Lin and Bercher 2022). The most widely used form of hierarchical planning is HTN planning, and we focus on the subset totally-ordered HTN (TO-HTN) planning. Lin and Bercher (2021) show that repairing a flawed TO-HTN model so that a given plan becomes valid is NP-complete, even with the insert-action-only variant. Lin and Bercher (2023) then extend these results to partially-ordered HTNs, identify three orthogonal NP-hardness sources, and demonstrate that action-insertion is the core driver of the complexity when the other two sources of hardness are fixed.

To date, only one documented HTN domain repair technique exists (Lin, Höller, and Bercher 2024), relying on elaborate re-encoding to transform the missing action repair problem into a new planning problem. However, general TO-HTN planning is EXPTIME-complete (Alford, Bercher, and Aha 2015), whereas the original TO-HTN repair task is only NP-complete. This gap motivates our contribution.

In this work, we propose a direct repair approach that leverages the increasing capability of LLMs to address missing action errors in TO-HTN domains. Like the previous technique, our approach takes as input a flawed domain and a plan known to be valid in a corresponding unflawed model. However, we utilise an LLM to solve a context-free grammar (CFG) repair problem derived from the TO-HTN planning domain. Based on the target plan, we prune the flawed domain into a more relevant subset by removing parts we deduce cannot be used to derive the target plan (an optimisation not performed by Lin, Höller, and Bercher (2024)). The LLM is prompted with both the filtered domain and the target plan, and it produces candidate action insertions that, when applied, render the plan derivable in both the filtered domain and the original domain, yielding a repaired domain which makes the target plan valid. Notably, we do not provide the LLM with any semantic cues such as action names or method names, making any direct recital of data in the training corpus untenable.

Related Work

The challenge of repairing flawed classical and hierarchical planning domains has been addressed from several angles in

the literature (Bercher, Sreedharan, and Vallati 2025). Here, we review prior work on domain repair in hierarchical planning, and discuss recent trends toward leveraging LLMs for automated planning and domain modelling.

Automated Domain Repair in Hierarchical Planning and Context-Free Grammars

Höller et al. (2014) showed that TO-HTN planning is structurally equivalent to CFGs, enabling grammar-theoretic analysis and repair methods (e.g., useless symbol removal (Hopcroft, Motwani, and Ullman 2001)). In the CFG setting, error-correction algorithms demonstrate how invalid input strings can be repaired via minimal edits to become part of the grammar's language. Aho and Peterson (1972) construct a covering grammar by adding "error productions" for insertions, deletions, and substitutions, and use an Earley-style parser augmented with error counts to recover both the list of edit operations and closest valid string in $O(n^3)$ time. Rajasekaran and Nicolae (2016) revisit the problem and use fast tropical matrix multiplication to yield sub-cubic time.

More recently, Barták et al. (2021) exploit the correspondence between TO-HTNs and CFGs to both verify and correct invalid plans via minimal action deletions. They cast verification as parsing an attribute grammar, prove that deciding whether k deletions suffice is NP-complete, and introduce a greedy parser that incrementally builds valid decompositions while tracking causal supports and constraint violations. Unlike pure validators, their approach returns a corrected plan with its decomposition tree, providing users with clear counterfactual explanations of which actions to remove to restore HTN validity.

In a similar vein, Lin, Höller, and Bercher (2024) also take an input plan and a flawed HTN model (in which the plan is not a solution), but instead of *repairing the plan* by deleting actions, they *repair the model* by inserting actions. Their approach recasts the model repair problem as a meta-HTN planning task whose solution yields the needed corrections to the original HTN model so that it can derive the input plan. To the best of our knowledge, no analogous classical CFG work directly repairs the grammar itself via automatic production rule updates. Raselimo and Fischer (2021) apply spectrum-based fault localisation to rank suspicious CFG rules and suggest patches, but they leave the actual grammar edits to the user rather than fully automating insertion and validation of new productions.

Complementing these approaches, Xiao et al. (2020) propose refining HTN methods through preference-guided task insertion. Their framework preserves existing domain knowledge by identifying and inserting missing subtasks into incomplete methods (similar to Lin, Höller, and Bercher (2024)), utilising prioritised preferences to determine the likelihood of method incompleteness.

LLMs for Planning Model Construction and Interactive Repair

Recent advances in LLMs have motivated research into their use for planning model construction and repair. Surveys such as the one by Pallagani et al. (2024) discuss a range of LLM applications in planning. Caglar et al. (2024) demonstrate that while LLMs can rapidly propose candidate model edits, integrating them with systematic search leads to more robust and verifiable domain repairs.

Another promising direction is the use of LLMs to assist with planning model construction and iterative repair. Guan et al. (2023) present a framework in which a pre-trained LLM is employed to generate an initial PDDL domain from natural language descriptions. By interfacing with validators and human feedback, the system iteratively refines the model until it is sound, mitigating many of the common pitfalls associated with LLM-generated outputs.

A recent development closely related to our focus is the use of LLMs to assist symbolic planners in open-world environments with incomplete domain models. Chen et al. (2024) introduce the Language-Augmented Symbolic Planner (LASP), where LLMs incrementally diagnose and repair planning models during execution by analysing errors, inferring missing knowledge, and updating the planning model as needed. This enables symbolic planners to recover from failures due to incomplete domain knowledge, combining the robustness of symbolic reasoning with the adaptability of LLMs for interactive model repair.

Oswald et al. (2024) investigate automating the generation of planning domain models from natural language using LLMs. Their framework translates textual descriptions into PDDL domain specifications and proposes automated evaluation metrics based on plan-set comparisons. By demonstrating that LLMs can generate domain models with a reasonable degree of correctness, their work directly addresses the challenge of manual domain modelling, aiming to lower the barrier to using automated planning by leveraging LLM-driven model construction.

Muñoz-Avila, Aha, and Rizzo (2025) propose ChatHTN, which explores using LLMs within a symbolic framework to generate action sequences in HTNs, but for the purpose of online planning instead of model repair. They augment their compound tasks with preconditions and effects, and if no suitable method decompositions are available, they use an LLM to propose an ad hoc decomposition.

In summary, LLMs have been utilised for various other hierarchical planning tasks and classical model construction and repair, but hierarchical repair has relied on systematic search or meta-reasoning.

Preliminaries

In this section, we review the foundational concepts on which our repair approach builds. We begin with classical planning, then introduce the formalism of TO-HTN planning, and finally recall the essentials of context-free grammars and their relationship to HTNs. These preliminaries set the stage for our CFG-based domain-repair procedure.

Classical Planning

A classical planning problem can be formally defined as a tuple $\langle F, A, I, G\rangle$ (Geffner and Bonet 2013), where:

- F is a finite set of fluents (state variables);
- $A \subseteq 2^F \times 2^F \times 2^F$ is a finite set of actions;

- $I \in 2^F$ is the initial state;
- $G \subseteq F$ is the set of goal conditions.

Each action $a \in A$ is characterised by its preconditions $pre(a) \subseteq F$, its add-effects $\mathit{eff}^+(a) \subseteq F$, and its delete-effects $\mathit{eff}^-(a) \subseteq F$. A planning problem consists of generating a sequence of actions, or *plan*, $\pi = \langle a_1, \ldots, a_m \rangle$ such that, starting from the initial state $s_0 = I$, each action a_i is applicable ($pre(a_i) \subseteq s_{i-1}$) and produces $s_i = \big(s_{i-1} \setminus \mathit{eff}^-(a_i)\big) \cup \mathit{eff}^+(a_i)$. We write $s \xrightarrow{a} s'$ to denote that applying action a in state s yields state s'. Thus a valid plan induces a state sequence

$$I = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_m} s_m,$$

where the final state satisfies the goal conditions, $s_m \supseteq G$.

TO-HTN Planning

A totally-ordered HTN (TO-HTN) planning problem is a tuple $\langle F, A, C, M, I, tn_I, G \rangle$, where

- F, A, I, G are as in classical planning;
- C is a finite set of *compound tasks*;
- M is a finite set of *decomposition methods*;
- tn_I is the *initial task network*.

The actions in A are called *primitive tasks* in hierarchical planning, and a *task network* is a finite sequence of tasks:

$$tn = \langle t_1, t_2, \ldots, t_n \rangle, \quad t_i \in A \cup C.$$

Compound tasks (also called *abstract tasks*) in C have no analogue in classical planning and require decomposition into sub-tasks to derive a plan π. Each $m \in M$ is a pair (c, tn_m), where $c \in C$ and $tn_m \in (A \cup C)^*$. A decomposition method (c, tn_m) decomposes a task network $tn = \langle t_1, \ldots, t_{k-1}, c, t_{k+1}, \ldots, t_n \rangle$ into a new task network $tn' = \langle t_1, \ldots, t_{k-1} \rangle + tn_m + \langle t_{k+1}, \ldots, t_n \rangle$, where $+$ denotes concatenation of sequences.

A *solution* to the TO-HTN problem is the resulting plan obtained via a sequence of method applications that fully decomposes tn_I into a primitive task network $tn_P = \langle a_1, \ldots, a_m \rangle$, $a_i \in A$ such that the induced action sequence forms a valid classical plan: that is, $I \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_m} s_m$, where $G \subseteq s_m$.

Context-Free Grammars

The structural parallels between context-free grammars (CFGs) and TO-HTN planning have been explored in the literature, such as for parsing to solve the plan verification problem (Höller et al. 2014; Barták, Maillard, and Cardoso 2018; Lin and Bercher 2022; Lin et al. 2023; Pantůčková and Barták 2023) or for plan repair (Barták et al. 2021), for problem compilations to solve TO-HTN problems (Höller 2024) or for theoretical investigations on the expressivity (Höller et al. 2014, 2016; Lin and Bercher 2021).

A CFG is a tuple (Hopcroft, Motwani, and Ullman 2001) $G = \langle \Gamma, \Sigma, P, S \rangle$, where:

- Γ is a finite set of non-terminals;
- Σ is a finite set of terminals;
- P is a finite set of production rules of the form $X \to \alpha$, with $X \in \Gamma$, $\alpha \in (\Gamma \cup \Sigma)^*$;
- $S \in \Gamma$ is the start symbol;
- The language of the grammar $L(G)$ is the set of terminal strings derivable from the start symbol, formally $L(G) = \{ w \in \Sigma^* \mid S \xrightarrow{*} w \}$, where $\xrightarrow{*}$ denotes zero or more applications of the production rules.

By mapping

$$\Gamma \leftrightarrow C, \quad \Sigma \leftrightarrow A, \quad P \leftrightarrow M', \quad S \leftrightarrow c_I,$$

and introducing a fresh start task c_I with accompanying method $m_I = (c_I, tn_I)$, and $M' = M \cup \{m_I\}$, any TO-HTN problem without facts can be viewed as a CFG, and producing a primitive task network is analogous to producing a string in the language of the CFG. This correspondence has been discussed in the literature (Höller et al. 2014) and underpins our CFG-based repair encoding. We could define the set of solutions of a TO-HTN problem as the language of its CFG intersected with solutions of its induced classical planning problem (obtained by ignoring the task hierarchy).

Domain Repair with Missing Actions

We now formally define the problem of repairing a flawed TO-HTN planning domain.

Let $\Pi^{\text{flaw}} = \big(D^{\text{flaw}}, s_I, tn_I, G\big)$ be a TO-HTN planning problem whose domain $D^{\text{flaw}} = (F, A, C, M^{\text{flaw}})$ contains decomposition methods that may be missing some primitive tasks. Furthermore, let $\pi = \langle a_1, \ldots, a_n \rangle$, $a_i \in A$ be an action sequence that the user expects to be derivable from tn_I. The goal of the domain-repair problem is to insert a (minimal) set of primitive tasks into the bodies of methods in M^{flaw} so as to obtain a repaired domain $D^* = \big(F, A, C, M^*\big)$ in which π can be constructed.

Atomic Correction. Let $m = (c, tn_m) \in M^{\text{flaw}}$ be a method with $tn_m = \langle t_1, \ldots, t_k \rangle$. For $a \in A$ and $0 \leq i \leq k+1$, define the atomic correction $I[a, m, i]$ as the insertion of action a at position i in tn_m, yielding an updated task network $\langle t_1, \ldots, t_{i-1}, a, t_i, \ldots, t_k \rangle$.

Correction Sequence. Let $\mathcal{O} = \langle o_1, \ldots, o_j \rangle$ be a sequence of atomic corrections, each of the form $I[a, m, i]$, applied in order, transforming M^{flaw} into a new set M^*.

Domain Repair Problem. Given a flawed TO-HTN planning problem $\Pi^{\text{flaw}} = (D^{\text{flaw}}, s_I, tn_I, G)$ and an action sequence π, the *domain repair problem* is to find an (optimal) correction sequence $\mathcal{O}$ such that, in the repaired problem $\Pi^* = (D^*, s_I, tn_I, G)$, π is derivable as a solution:

$$\text{Find (minimal) } \mathcal{O} \text{ such that } \pi \text{ is a solution to } \Pi^*.$$

Optimality is typically measured by the number of action insertions into the methods, i.e., the minimal number of corrections $|\mathcal{O}|$ required to make π derivable, noting that there's no strict requirement for $D^* = D$.

Note that according to this definition, all state features become irrelevant since no action insertion into the model can make a given non-executable action sequence executable. This thus renders the TO-HTN repair problem semantically equivalent to a CFG-repair problem.

Repair Approach

Our approach repairs flawed TO-HTN models represented as CFGs. Given as input a flawed domain D^{flaw} and a target plan π, our method produces a repaired domain D^* in which π is derivable. The process consists of three main stages:

1. *Structural Filtering and CFG Pruning*: We systematically remove methods, non-terminals, and terminals that cannot participate in any derivation of the target plan, producing a much smaller, focused CFG that retains only the domain's relevant structure.
2. *Masked Prompt Construction and LLM-Based Repair*: The pruned and abstracted grammar, along with the masked target plan, are encoded into a prompt for the LLM, which is tasked with proposing insertions of missing actions to enable derivation of the plan.
3. *Automated Plan Verification*: The repaired CFG is automatically parsed and then checked using a CFG membership test to confirm that the plan is indeed derivable in the repaired model.

While one might consider simply presenting the full flawed domain and target plan to an LLM and asking it to suggest repairs, this naive strategy is both impractical and likely to be ineffective. Unfiltered planning domains typically contain a substantial number of irrelevant non-terminals, terminals, and production rules that cannot participate in any derivation of the target plan, regardless of which repairs are applied. Including these extraneous elements in the LLM prompt would significantly inflate the size and complexity of the prompt, exceeding the size of the LLM's context window in the worst case and in the best case still slowing inference while increasing cost. More subtly, such irrelevant or unreachable rules may distract the LLM, leading it to propose "hallucinated" repairs that are unrelated to the goal, or to miss relevant opportunities for repair entirely. Therefore, our pipeline begins with a dedicated structural filtering step that aggressively prunes the flawed CFG down to a more relevant subset for deriving the target plan.

Structural Filtering and CFG Pruning

Our algorithm is inspired by how the so-called task decomposition graph (TDG) is used to restrict the domain for parts that could contribute to any solution, described in the context of grounding (Behnke et al. 2020). We extend it to prune the planning domain with respect to a specific target plan. Given a flawed TO-HTN domain (represented as a CFG) and a target plan $\pi = \langle a_1, \ldots, a_L \rangle$, the first stage prunes the grammar to extract only those non-terminals, terminals, and production rules potentially participating in the derivation of π. This reduction is essential to ensure the prompt fits within the LLM's context window, while preserving all information necessary for possible plan reconstruction.

We define $\infty := L + 1$ as a value greater than any achievable plan length. Each production rule r is annotated with a Boolean flag `usable` that is set to true only if r could potentially be used to derive the target plan.

The structural filtering proceeds as follows:

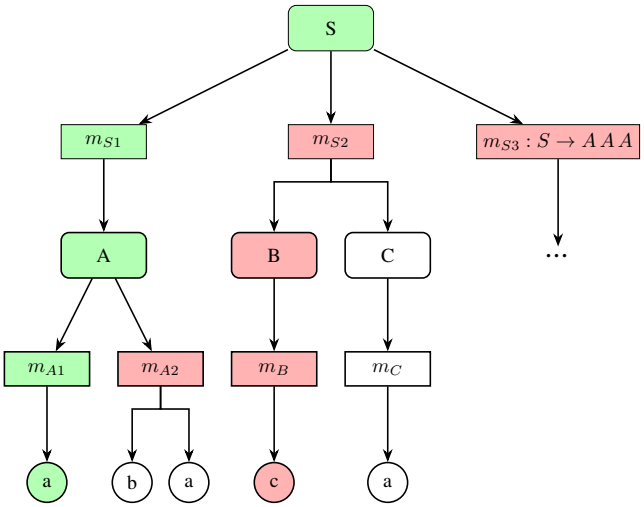

Figure 1: Pruning for plan $\pi = \langle a, b \rangle$ to produce a reduced subset of the original CFG. The circles are terminal symbols, the rounded boxes are non-terminal symbols, the rectangular boxes are the production rules, and the edges link the production rules to symbols. Red components have been invalidated and green components are the new pruned CFG.

1. **Bottom-up minimal-length computation:** For each terminal $a \in \Sigma$, set $\texttt{min\_len}[a] := 1$. For each non-terminal $c \in \Gamma$, initialize $\texttt{min\_len}[c] := \infty$. Then, iteratively update $\texttt{min\_len}[c]$ for all c as follows: For each production rule $r : c \to X_1, \ldots, X_k$, if all X_i are either (a) terminals present in π, or (b) non-terminals with $\texttt{min\_len}[X_i] < \infty$, and the total length does not exceed L, mark r as usable and update $\texttt{min\_len}[c]$ accordingly. Repeat this process to a fixed point (as in dynamic programming).
2. **Top-down reachability:** Starting from the initial symbol S, perform a breadth-first search (BFS), visiting only non-terminals and production rules marked as usable. Enqueue every non-terminal reachable from S via usable productions. This identifies the subset of symbols and rules relevant to deriving π.
3. **Pruning:** Retain only production rules r that are both usable and whose left-hand side is reachable. All unreachable non-terminals and terminals not present on the right-hand side of any retained rule are pruned from the CFG.

Figure 1 depicts a simple example of a CFG being pruned given the target plan $\pi = \langle a, b \rangle$. Beginning with the leaf nodes from left to right, we can see that a, the child of m_{A1}, does not directly violate π, so it is not ruled out. Neither do b or a, the children of m_{A2}. However, m_{A2} itself does violate π due to lacking a valid decomposition matching any subsequence of π, thus it cannot produce π under any correction $\mathcal{O}$, allowing its exclusion. Towards the centre we have terminal symbol c which is invalid because it is not a subsequence of π (as it does not appear in π). Furthermore, this constraint violation propagates up to m_B, and subsequently to B as it has no other applicable production rule. Likewise, m_{S2} is

Listing 1: Excerpt from a masked LLM prompt file.

```
I have a context free grammar (CFG). The
    initial symbol is 'A1'.
<Additional instructions and details>
Production rules:
rule_1: A1 -> A14 A11 A2
rule_2: A2 -> 1
rule_3: A2 -> A7 A6 A5 A3
...
Target sequence:
1 6 7 4 5 8 9 1 1 10 13 1 1 1 12 ...
```

also invalid because it necessarily contains B. Non-terminal symbol C now becomes orphaned, despite all its child nodes being valid, because it does not have a valid parent and now cannot be reached from the starting symbol S. Lastly, on the right-most portion of this figure, we can see that m_{S3} is also invalid because although its children are all valid, the minimum possible length of any plan it can produce is greater than $|\pi|$.

Masked Prompt Construction and LLM-Based Repair

After filtering, we construct a prompt file for the LLM:

- All identifiers are masked. Primitive tasks become integers; non-terminals are labeled as $A_1, A_2, \ldots$; production rules are labeled as $\text{rule}_1, \text{rule}_2, \ldots$.
- The prompt lists terminal and non-terminal symbols, the initial symbol, the production rules, and the target plan.
- The instructions specify that the only allowed repair operation is the *insertion* of terminal symbols into existing production rules; deletion, reordering, or new rule creation is prohibited. Additionally, the LLM is asked to minimise the total number of insertions required.

Crucially, the prompt provides *no information about action preconditions or effects*, only the grammar structure and the abstract plan. All symbol names and domain semantics are masked. As a result, the LLM must rely entirely on symbolic manipulation and structural reasoning; it cannot exploit semantic cues or original action names. This also has a benefit of making the prompt shorter to fit within the context length of an LLM. Although HTN domains may appear in LLM pre-training data, our CFG masking prevents memorisation or direct recitation.

Prompt File Example: Below is a representative excerpt from an actual prompt file provided to the LLM. The full prompt for this instance is included in the appendix and is available at 10.5281/zenodo.17620754 along with our entire code base.

This prompt format ensures that the LLM is presented only with abstract, structural information, with no possibility to exploit naming or semantics. The LLM's output is parsed automatically to extract only the rule modifications, as specified by the required output format.

Automated Plan Verification

After parsing and applying LLM insertions, we verify the repaired model's derivability of the target plan. This verification reduces to the standard CFG membership problem. Specifically, we first binarise all productions by converting arbitrary-arity rules to binary rules in *Chomsky Normal Form (CNF)* (Hopcroft, Motwani, and Ullman 2001). To briefly recap, in CNF, every production is either of the form $X \rightarrow YZ$ (where Y and Z are non-terminals), or $X \rightarrow a$ (where a is a terminal). We then apply the classical Cocke-Younger-Kasami (CYK) parsing algorithm to determine whether the sequence of terminal symbols corresponding to the target plan is generated by the grammar.

Experimental Evaluation

We conduct our experiments using the same benchmark set introduced by Lin, Höller, and Bercher (2024). Each domain repair instance in this benchmark set was generated by randomly removing primitive tasks from the method bodies of the IPC 2020 HTN planning benchmarks with a 30% probability. We evaluate four distinct repair methods:

1. The original optimal symbolic encoding (**Enc***) using A* search using landmark-cut heuristics.
2. Symbolic encoding ($\widehat{\textbf{Enc}}$), using (suboptimal) Greedy A* (weight=5) with add heuristics.
3. Our primary LLM-based approach (**LLM**), described earlier, using OpenAI's model `o4-mini-2025-04-16`.
4. Another LLM-based variant (**GPT-OSS**) using open-weight model `openai/gpt-oss-120b`.

Unless otherwise indicated, the references we make to our LLM-based approach will reference the model `o4-mini-2025-04-16`. We note that the LLM-based approach does not guarantee optimality, so we primarily focus on comparing it to the suboptimal encoding method using a much faster heuristic for a more balanced comparison. We run experiments on an Apple M4 Pro with 64GB RAM using MacOS 15.5 and show the results for the optimal encoding (Lin, Höller, and Bercher 2024), our suboptimal encoding, and both LLM variants. We chose to use the add heuristic for the suboptimal encoding since it's shown to be the best performing in the literature (Höller and Behnke 2021). In line with the previous method, we set a total time limit of 20 mins. LLM API calls were set using `reasoning_effort:=high` and `max_completion_tokens:=65536`. LLM responses were classified as failures if they contained unparseable outputs, invalid updates, or failed to derive the target plan.

As seen in Table 1, both LLM variants dramatically outperform the symbolic encodings in terms of raw coverage, with o4-mini solving 145 instances and GPT-OSS solving 140, compared to 76 and 57 for the suboptimal and optimal encodings respectively. The o4-mini model yields the best overall coverage, while the GPT-OSS model closely tracks its performance and even attains the highest coverage in the Monroe (PO) domain (19/19 solved). o4-mini failed in only 11 cases: two unsanctioned repairs and nine failed plan

Domain	Total	Enc*	$\widehat{\text{Enc}}$	o4-mini	oss
Hiking	20	0	11	**17**	15
Transport	18	0	10	**17**	16
Entertainment	10	**10**	**10**	**10**	**10**
Rover	20	13	14	**18**	17
Monroe (FO)	17	8	0	**17**	16
Depots	20	4	3	**17**	**17**
Woodworking	4	3	**4**	**4**	**4**
Satellite	13	10	11	**13**	12
Blocksworld	5	1	0	**5**	**5**
Monroe (PO)	19	5	3	18	**19**
Childsnack	10	3	**10**	9	9
Total	156	57	76	**145**	140

Table 1: Comparison of solved instances across all methods.

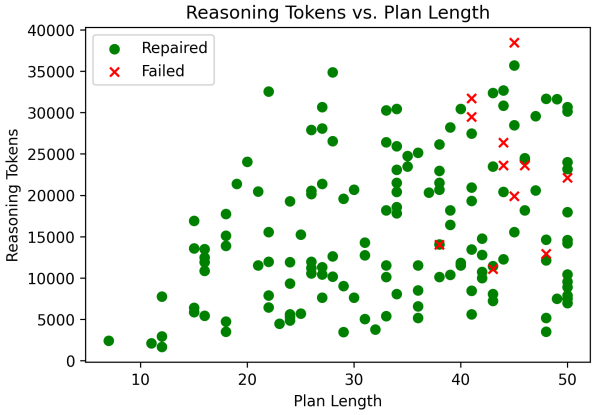

Figure 2: Scatterplot of LLM reasoning token count versus plan length.

derivations. In the original paper, the benchmark was created by running a primitive task removal method across roughly 200 problems, but only modified 167 instances. However, here we further restricted these modified instances to exclude ones that were not altered enough to render the accompanying target plan invalid, yielding only 156 instances. If we wanted to include these trivial repairs ($\mathcal{O} = \emptyset$), we would see that the original optimal encoder (Enc*) scored 62/167 on this slightly expanded problem set, solving only 5/11 of these empty action insertion repair problems.

However, this is not a direct comparison as the LLM approach requires more advanced hardware. Figure 2 plots the number of tokens used by the LLM versus the plan length for each repair instance. We observe a clear trend indicating that failures become more frequent as both plan length and token usage increase. Nevertheless, the LLM successfully repairs many longer instances, demonstrating strong symbolic reasoning capabilities. However, the failure rate notably rises for plans exceeding approximately 40 steps or as token counts approach the LLM's context window limits. This emphasises the necessity for effective domain pruning and suggests future improvements such as chunked or iterative prompting to manage larger and more complex in-

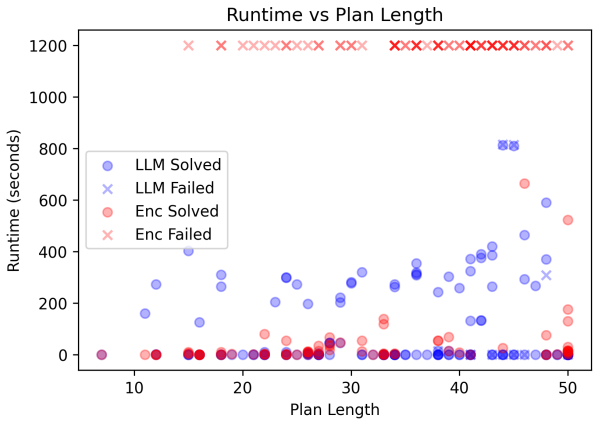

Figure 3: Runtime comparison vs. plan length.

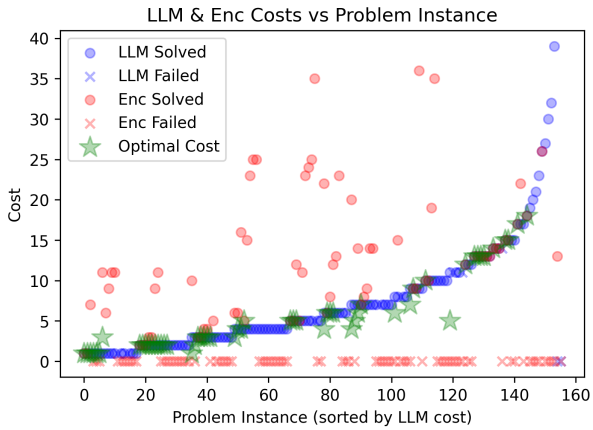

Figure 4: Repair costs per problem instance sorted by LLM cost. Optimal costs (green stars) are available for instances solved by Enc*. LLM solutions (blue circles) and suboptimal encoding (red circles) are compared to these optima.

stances effectively. Logistic regression analysis further reveals that plan length itself is the primary predictor of failure (plan length: $p = 0.015$; token count: $p = 0.052$), reinforcing the idea that structural complexity, rather than prompt size constraints, fundamentally limits LLM-based repair performance.

Figure 3 visualises the runtime characteristics of the LLM and suboptimal encoding methods. The encoding failures (red crosses) represent instances where exponential blow-ups led to timeout failures at 1200 s, particularly for complex instances. In contrast, LLM failures (blue crosses) are never due to timeouts but rather incorrect solution attempts. Successful encoding repairs are very fast, while the pruning and verification gradually scales with plan length.

Figure 4 compares repair insertion costs, highlighting the near-optimal quality of the LLM repairs. Among the 57 instances with known optimal costs, the LLM method achieves exact optimality in 42 cases (74%) and incurs only 6% higher average cost overall. In cases where the LLM ap-

proach is suboptimal, it increases insertion cost by 37% on average. In contrast, the greedy suboptimal encoding shows much greater deviation; 44% higher average cost overall, ballooning up to 192% above optimal when excluding optimal instances. For LLM repairs across the entire benchmark, the number insertions into each updated method is often small (mean=2.06, std=2.63, median=1), suggesting balanced repair and not merely the exploitation of mass insertions into a single method. These evaluations demonstrate the effectiveness of our LLM-based repair approach: it maintains a robust coverage across diverse problems, scales predictably with increasing complexity, and reliably delivers repairs close to optimal, significantly surpassing the faster but highly variable suboptimal encoding method. Nonetheless, the occasional invalid LLM outputs emphasise a critical area for future work, including potential integrations of lightweight symbolic verification or incremental refinement to boost solution quality and reliability.

Discussion

Our experimental results clearly demonstrate the strengths and trade-offs of our approach compared to the existing implemented symbolic encoding method, highlighting critical considerations such as coverage, robustness, runtime scalability, and optimality. A key advantage is its much higher coverage (93%), dramatically outperforming both the optimal encoding method (37%) and the suboptimal-heuristic-based encoding method (49%). This improvement is especially pronounced in challenging domains such as Monroe, Depots, Hiking, and Transport, where symbolic methods struggled with exponential blow-ups and timed out. A critical factor supporting the effectiveness of our approach is the inherent structural equivalence between grounded TO-HTN planning problems and CFGs, with our work both enhancing TO-HTN repair and also making a secondary contribution to a novel area of CFG repair. By exploiting this equivalence, we can prune and utilise direct, grammar-based verification methods and avoid additional complexity introduced by symbolic encodings. This proved successful and we did not require exploring more elaborate inference patterns to pursue even better results from our usage of LLMs.

The symbolic encoding method demonstrates a sharp dichotomy in runtime scalability, solving simpler instances rapidly but failing due to exponential explosion on larger and more complex problems. In contrast, our approach scales more predictably and gradually with plan length, not approaching the imposed timeout or token limit, making it more suitable for practical applications where high coverage across diverse problem instances is essential. Potential portfolio solutions could be considered which first utilise the optimal encoding and then fall back to an LLM approach.

However, a key limitation of our approach remains the absence of formal optimality guarantees due to the way we use LLMs. Although our experiments show that the method consistently produces near-optimal results, the absence of rigorous guarantees can be problematic for certain critical applications. Addressing this limitation will be an important aspect of future research. The most immediate suggestion would be to combine our pruning with the previous encoding method. Other potential solutions include integrating lightweight verification steps, such as local search or iterative refinements using symbolic optimisation techniques. A particularly promising avenue is the integration of LLMs with automated symbolic repair methods into iterative frameworks. This hybrid approach could leverage the broad coverage and intuitive reasoning of LLMs while still providing strong formal guarantees and optimality.

While we focused only on missing primitive actions, real-world domains often contain multiple error types. Future work should explore extending our approach to other error classes, including deletions, incorrect task orderings, missing methods, or errors related to lifted representations such as incorrect typing or incorrect argument arities. We could also allow multiple valid plans and multiple invalid plans to further guide and constrain the repair. The inherent flexibility and adaptability of LLM-based reasoning suggest significant potential advantages over purely symbolic optimisation approaches, which may require entirely new encodings or extensive engineering effort to handle each novel error type.

Finally, an exciting and largely unexplored potential of LLM-based domain repair is the incorporation of external common-sense knowledge. Symbolic methods traditionally operate strictly within the knowledge encoded in the domain model. However, LLMs inherently possess broad general knowledge, which could be leveraged to repair domain errors requiring common-sense reasoning beyond the provided domain specification. For instance, in realistic scenarios involving incomplete or ambiguous models, an LLM could infer missing actions or constraints based on general world knowledge (Behnke et al. 2015), enhancing not just the correctness but also the semantic richness and practical utility of automatically repaired planning domains.

Conclusion

We have introduced a novel approach leveraging LLMs for repairing TO-HTN planning domains by directly inserting missing primitive actions. By exploiting the inherent structural equivalence between grounded TO-HTNs and CFGs, we significantly simplify and accelerate the repair process, filtering our search space and avoiding the complexity and exponential blow-ups commonly experienced by symbolic re-encoding methods. Extensive empirical evaluation demonstrates that our LLM-based method substantially outperforms both optimal and suboptimal symbolic encodings, achieving dramatically higher coverage and more predictable scalability, while consistently producing repairs near the optimal insertion cost. Our work can serve as a basis for a multitude of useful extensions that would be straightforward to achieve, yet would be presumably much more difficult in a pure optimisation framework.

Acknowledgments

Pascal Bercher is the recipient of an Australian Research Council (ARC) Discovery Early Career Researcher Award (DECRA), project number DE240101245, funded by the Australian Government.

References

Aho, A. V.; and Peterson, T. G. 1972. A minimum distance error-correcting parser for context-free languages. *SIAM Journal on Computing*, 1(4): 305–312.

Alford, R.; Bercher, P.; and Aha, D. 2015. Tight Bounds for HTN Planning. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 7–15. AAAI Press.

Alford, R.; Shivashankar, V.; Roberts, M.; Frank, J.; and Aha, D. W. 2016. Hierarchical Planning: Relating Task and Goal Decomposition with Task Sharing. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, 3022–3029. IJCAI.

Barták, R.; Maillard, A.; and Cardoso, R. 2018. Validation of Hierarchical Plans via Parsing of Attribute Grammars. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS 2018)*, 11–19. AAAI Press.

Barták, R.; Ondrčková, S.; Behnke, G.; and Bercher, P. 2021. Correcting Hierarchical Plans by Action Deletion. In *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning (KR 2021)*, 99–109. IJCAI.

Behnke, G.; Höller, D.; Schmid, A.; Bercher, P.; and Biundo, S. 2020. On Succinct Groundings of HTN Planning Problems. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI 2020)*, 9775–9784. AAAI Press.

Behnke, G.; Ponomaryov, D.; Schiller, M.; Bercher, P.; Nothdurft, F.; Glimm, B.; and Biundo, S. 2015. Coherence Across Components in Cognitive Systems – One Ontology to Rule Them All. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, 1442–1449. AAAI Press.

Bercher, P.; Alford, R.; and Höller, D. 2019. A Survey on Hierarchical Planning – One Abstract Idea, Many Concrete Realizations. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, 6267–6275. IJCAI.

Bercher, P.; Sreedharan, S.; and Vallati, M. 2025. A Survey on Model Repair in AI Planning. In *Proceedings of the 34th International Joint Conference on Artificial Intelligence (IJCAI 2025)*, 10371–10380. IJCAI.

Caglar, T.; Belhaj, S.; Chakraborty, T.; Katz, M.; and Sreedharan, S. 2024. Can LLMs Fix Issues with Reasoning Models? Towards More Likely Models for AI Planning. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI 2024)*, 20061–20069. AAAI Press.

Chen, G.; Yang, L.; Jia, R.; Hu, Z.; Chen, Y.; Zhang, W.; Wang, W.; and Pan, J. 2024. Language-Augmented Symbolic Planner for Open-World Task Planning. In *Proceedings of Robotics: Science and Systems*. Delft, Netherlands.

Erol, K.; Hendler, J.; and Nau, D. S. 1996. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence*, 18(1): 69–93.

Geffner, H.; and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Springer International Publishing.

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 *In Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS 2023)*, 79081–79094. Neural Information Processing Systems Foundation.

Höller, D. 2024. The Toad System for Totally Ordered HTN Planning. *Journal of Artificial Intelligence Research (JAIR)*, 80: 613–663.

Höller, D.; and Behnke, G. 2021. Loop Detection in the PANDA Planning System. In *Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS 2021)*, 168–173. AAAI Press.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2014. *Language Classification of Hierarchical Planning Problems*, 447–452. IOS Press.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2016. Assessing the Expressivity of Planning Formalisms through the Comparison to Formal Languages. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS 2016)*, 158–165. AAAI Press.

Hopcroft, J. E.; Motwani, R.; and Ullman, J. D. 2001. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2nd edition edition.

Langley, P. 2025. Learning Hierarchical Task Knowledge for Planning. In *Proceedings of the 39th AAAI Conference on Artificial Intelligence (AAAI 2025)*, 28652–28656. AAAI Press.

Lin, S.; Behnke, G.; Ondrčková, S.; Barták, R.; and Bercher, P. 2023. On Total-Order HTN Plan Verification with Method Preconditions – An Extension of the CYK Parsing Algorithm. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI 2023)*, 12041–12048. AAAI Press.

Lin, S.; and Bercher, P. 2021. Change the World – How Hard Can that Be? On the Computational Complexity of Fixing Planning Models. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI 2021)*, 4152–4159. IJCAI.

Lin, S.; and Bercher, P. 2022. On the Expressive Power of Planning Formalisms in Conjunction with LTL. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS 2022)*, 231–240. AAAI Press.

Lin, S.; and Bercher, P. 2023. Was Fixing this *Really* That Hard? On the Complexity of Correcting HTN Domains. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI 2023)*, 12032–12040. AAAI Press.

Lin, S.; Höller, D.; and Bercher, P. 2024. Modeling Assistance for Hierarchical Planning: An Approach for Correcting Hierarchical Domains with Missing Actions. In *Proceedings of the 17th International Symposium on Combinatorial Search (SoCS 2024)*, 55–63. AAAI Press.

McCluskey, T. L.; Richardson, N. E.; and Simpson, R. M. 2002. An Interactive Method for Inducing Operator Descriptions. In *Proceedings of the 6th International Conference on*

Artificial Intelligence Planning Systems (AIPS 2002), 121–130. AAAI Press.

Muñoz-Avila, H.; Aha, D. W.; and Rizzo, P. 2025. ChatHTN: Interleaving Approximate (LLM) and Symbolic HTN Planning. In *Proceedings of the 2nd International Conference on Neuro-symbolic Systems (NeuS 2025)*, 1–13. PMLR.

Oswald, J.; Srinivas, K.; Kokel, H.; Lee, J.; Katz, M.; and Sohrabi, S. 2024. Large Language Models as Planning Domain Generators. In *Proceedings of the 34th International Conference on Automated Planning and Scheduling (ICAPS 2024)*, 423–431. AAAI Press.

Pallagani, V.; Muppasani, B. C.; Roy, K.; Fabiano, F.; Loreggia, A.; Murugesan, K.; Srivastava, B.; Rossi, F.; Horesh, L.; and Sheth, A. 2024. On the Prospects of Incorporating Large Language Models (LLMs) in Automated Planning and Scheduling (APS). In *Proceedings of the 34th International Conference on Automated Planning and Scheduling (ICAPS 2024)*, 432–444. AAAI Press.

Pantůčková, K.; and Barták, R. 2023. Using Earley Parser for Recognizing Totally Ordered Hierarchical Plans. In *Proceedings of the 26th European Conference on Artificial Intelligence (ECAI 2023)*, 1819–1826. IOS Press.

Rajasekaran, S.; and Nicolae, M. 2016. An error correcting parser for context free grammars that takes less than cubic time. In *Language and Automata Theory and Applications*, 533–546. Springer.

Raselimo, M.; and Fischer, B. 2021. Automatic grammar repair. In *Proceedings of the 14th International Conference on Software Language Engineering*, 126–142. ACM.

Xiao, Z.; Wan, H.; Zhuo, H. H.; Herzig, A.; Perrussel, L.; and Chen, P. 2020. Refining HTN Methods via Task Insertion with Preferences. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI 2020)*, 10009–10016. AAAI Press.