

Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Master Thesis

LLMs for AI Planning: A Study on Error Detection and Correction in PDDL Domain Models

Kshitij Patil

Course of Study: Computer Science

Examiner: Dr. Ilche Georgievski

Supervisor: Dr. Ilche Georgievski

Commenced: May 1, 2024

Completed: November 4, 2024

Abstract

Domain modeling is fundamental to Artificial Intelligence (AI) planning, since it facilitates the characterization and organization of intricate planning challenges. A significant difficulty in domain modeling is to guarantee that these models are accurate and comprehensive, encompassing not just syntactic regulations but also complex semantic inter dependencies. Deficiencies in domain models—such as duplicate effects, unmet preconditions, or absent dependencies—can significantly affect planning results, highlighting the necessity for effective error detection and repair systems to improve the efficacy of AI planning.

This research examines the capability of Large Language Models (LLMs) to enhance the precision and resilience of domain models by detecting and rectifying flaws intrinsic to these planning frameworks. Due to the intricacy of domain models in AI planning, LLMs present a viable method for automating error detection and refinement, tackling particular challenges such as complementary effects, invalid parameters, and immutable predicates, which are essential for precise planning domain specifications. The research entailed systematic evaluation and progressive refinement of Planning Domain Definition Language (PDDL) models utilizing GPT-4 to analyze the efficacy of LLMs in diminishing syntactic and semantic error rates, therefore enhancing the reliability and quality of PDDL Domain models in AI planning.

The results indicate that GPT-4 exhibits some ability to identify and rectify certain mistake types; nevertheless, its performance is not consistently dependable. Semantic mistakes, particularly those involving intricate logical dependencies and complicated interrelations among model components, continue to pose significant challenges for LLMs to address successfully. The LLM-based technique has potential in addressing minor faults but encounters difficulties with complex domain-specific problems, resulting in only marginal enhancements in model accuracy. These findings suggest that, despite their capabilities, LLMs now do not achieve complete automation of PDDL domain model refinement and need considerable manual supervision.

Although LLMs offer a potential foundation for improving PDDL models, their existing capabilities are constrained in tackling intricate semantic problems that need sophisticated reasoning. This work emphasizes the accomplishments of LLMs in mistake identification and repair, while also pinpointing areas for future research to further their application in AI planning.

Contents

1	Introduction	15
1.1	Importance of AI Planning	15
1.2	Motivation	15
1.3	Overview of the State of the Art	16
1.4	Gaps in the State of the Art	17
1.5	Research Question	18
1.6	Methodology	18
1.7	Original Contribution	19
1.8	Structure	19
2	Background Information	21
2.1	AI Planning	21
2.2	Planning Domain Definition Language	21
2.3	Common Errors in PDDL Domain Files	23
2.4	Why Errors in PDDL Matter	24
2.5	Error Detection and Evaluation Tools	24
2.6	Large Language Model (LLM) and Prompting	25
3	Related Work	27
4	Proposed Framework	31
4.1	Training Stage	31
4.2	Testing Stage	33
4.3	Evaluation Stage	36
5	Implementation	39
5.1	Training Dataset	39
5.2	Choice of LLM	39
5.3	Testing Dataset	43
5.4	Domain Error Analyzing Tool	44
5.5	LLM Integration for Error Correction	48
5.6	AI Planner and Plan Evaluation	52
6	Experimental Evaluation	55
6.1	Experimental Setup	55
6.2	Results	57
7	Conclusion	71
	Bibliography	73

A Appendix	77
A.1 Plan Evaluation Process	77

List of Figures

4.1	Training Architecture	31
4.2	Testing Architecture	33
4.3	Design and objective spaces for a two-objectives planning problem [KSV+13]	38
5.1	Creating new pddl_proj file	46
5.2	UI depicting uploaded Domain files	46
5.3	UI highlighting Error Log	47
5.4	Pre- and Post-LLM Enhanced Domain files	51
6.1	Example of an errorless PDDL Domain	57
6.2	Example of Parking PDDL Domain	59
6.3	Example of pegsolitaire PDDL Domain	59
6.4	Error log for an Incomplete Domain Model	60
6.5	Example of 'Pathways' Domain Model	60
6.6	Pre- and Post-LLM Enhanced Error trend	67
6.7	Parking domain Pre-LLM Enhancement metric visualization	68
6.8	Parking domain Post-LLM Enhancement metric visualization	68
6.9	pegsolitaire domain Pre-LLM Enhancement metric visualization	69
6.10	pegsolitaire domain Post-LLM Enhancement metric visualization	69
A.1	VS-Code PDDL Overview Page	77
A.2	Uploading a Domain and its corresponding Problem file	78
A.3	Selecting TFD-Planner to generate valid plan	78
A.4	Generated Plan	79
A.5	Plan Evaluation using VAL	79

List of Tables

5.1	Comparison of Features between GPT-3, GPT-4, and LLaMa	40
5.2	Comparison of Knowledge Engineering Tools and Their Features [SK20]	45
6.1	Evaluation Metrics for Planning Domain Definition Language (PDDL) Domain Quality Assessment	57
6.2	Semantic and Syntactic Errors Pre and Post LLM-Enhancement	61
6.3	Semantic and Syntactic Errors Pre and Post LLM-Enhancement (ipc-2002)	62
6.4	Semantic and Syntactic Errors Pre and Post LLM-Enhancement (ipc-2004)	62
6.5	Semantic and Syntactic Errors Pre and Post LLM-Enhancement (ipc-2006)	63
6.6	Semantic and Syntactic Errors Pre and Post LLM-Enhancement (ipc-2008)	63
6.7	Semantic and Syntactic Errors Pre and Post LLM-Enhancement (ipc-2011)	64
6.8	Semantic and Syntactic Errors Pre and Post LLM-Enhancement (ipc-2014)	64
6.9	EPCS Metrics for PDDL Domain Files	65
6.10	Evaluation of the generated plan Pre- and Post-LLM Enhancement	70

List of Listings

2.1	PDDL Domain for Blocksworld	22
2.2	PDDL Problem for Blocksworld	23

List of Abbreviations

AI Artificial Intelligence. 15

CoT Chain-of-Thought. 18

EPCS Enhance PDDL Correctness Score. 19

IPC International Planning Competition. 19

LLM Large Language Model. 15

PDDL Planning Domain Definition Language. 9

TFD Temporal Fast Downward. 19

1 Introduction

1.1 Importance of AI Planning

Artificial Intelligence (AI) has seen substantial growth in the past few decades, transforming industries from healthcare and finance to transportation and robotics. Within AI, AI planning plays a crucial role in automating decision-making processes, helping systems to generate sequences of actions to achieve specific goals. Whether it is a robotic arm in a manufacturing plant that needs to pick and place objects efficiently, or a self-driving car that must navigate complex traffic scenarios, AI planning is the backbone that enables these systems to act autonomously and effectively [Geo23].

In AI planning, constructing domain models is essential for representing the knowledge and structure needed to solve complex problems. These models define the actions, predicates, and their relationships within the domain, using formal languages such as the Planning Domain Definition Language (PDDL) [AHK+98]. PDDL allows AI planners to understand the possible actions that can be taken, the preconditions for these actions, and the expected outcomes [McC03]. However, constructing PDDL domain models is often a labor-intensive, error-prone process that requires significant planning expertise. Errors, whether syntactic or semantic, can severely hinder the planner's ability to generate valid or optimal plans, making domain modeling one of the key bottlenecks in scaling AI planning solutions to real-world applications [Geo25].

This research investigates the potential of Large Language Models (LLMs) in automating and enhancing the refinement of PDDL domain models. The ultimate goal is to reduce the manual effort involved in model creation while ensuring that the resulting models are both syntactically correct and semantically sound, thereby facilitating more efficient and reliable AI planning.

1.2 Motivation

The impetus for employing LLMs in domain modeling arises from recent advancements in natural language processing (NLP). LLMs, such as GPT-4, have exhibited exceptional proficiency in producing structured outputs, including computer code, knowledge graphs, and process models, from natural language inputs. These models are trained on extensive text data and may produce contextually relevant replies, even in specialized languages such as PDDL [XYJ+23].

Considering the intricacies and formal stipulations of domain modeling in AI planning, utilizing LLMs to enhance PDDL models presents significant promise. This methodology may optimize the domain modeling process, enabling specialists to focus on overarching planning objectives, while LLMs handle the complex aspects of model validation and error detection [LJZ+23]. This corresponds with previous

research about the viability of employing LLMs to enhance structured problem-solving tasks in domain-specific languages such as PDDL, highlighting their effectiveness in automated planning frameworks [SDS+24].

The effectiveness of automated planners largely depends on the organization and quality of the foundational domain models. Vallati et al. (2020) highlight that the configuration of domain models—specifically, the organization of components like operators and predicates—can significantly influence planner performance. Domain-specific optimized configurations can augment plan generation efficiency by minimizing runtime and enhancing the planner’s ability to address intricate, domain-specific needs [VCMH21]. This rationale highlights the significance of LLMs as instruments for enhancing PDDL models, with the objective of achieving efficiencies and minimizing errors related to both syntax and semantics.

The preliminary use of LLMs in the generation of PDDL models has demonstrated potential; nevertheless, the optimization and augmentation of current models remain mostly unexamined. Domain models created by humans frequently exhibit incompleteness or flaws, which may vary from basic syntactic problems to profound semantic inconsistencies [SB23]. These flaws can hinder AI planners from implementing legitimate plans or result in inferior solutions. Consequently, the capacity to utilize LLMs for error detection and correction in domain models might markedly improve the scalability and reliability of AI planning systems.

1.3 Overview of the State of the Art

The current state of the art in PDDL model generation and validation is primarily focused on human-driven efforts, supported by automated tools for error detection. PDDL Studio and VAL are two commonly used tools in the field. PDDL Studio is designed to help users create and validate PDDL models by identifying both syntactic and semantic errors [SB23]. It checks for inconsistencies in action definitions, preconditions, and effects, flagging potential issues that could lead to faulty plans. VAL, on the other hand, validates the logical consistency of the domain models by running simulations to ensure that plans generated from the models are executable.

Recent research has begun exploring the potential of LLMs for generating structured outputs like programming code and domain-specific languages. Studies have shown that LLMs can generate PDDL-like models from structured natural language inputs, but these models are often fraught with errors. Research by Guan et al. (2023) and Pallagani et al. (2024) has demonstrated that LLMs can assist in constructing world models and task planning frameworks [GVSK23][PMR+24], but these efforts are still in their infancy when it comes to generating error-free, refined models ¹.

The key challenge lies in the semantic validation of PDDL models. While syntactic errors, such as missing parentheses or incorrect keywords, are relatively easy for automated tools or syntax-checking software to detect and fix, semantic errors present a far greater challenge [LJZ+23]. These errors arise from inconsistencies in the logical relationships between actions, predicates, and their parameters, which can result in invalid or suboptimal plans. For instance, an action may contain unsatisfied preconditions that make it impossible to execute within the given domain, or the effects of an action might contradict the

¹[semantic-error-chapter-80-understanding-detecting](#)

preconditions of subsequent actions, creating logical conflicts that hinder planning outcomes [VCMH21]. Tools that can interpret the intricate relationships inside domain models are needed to address these semantic problems, and LLMs are currently being studied for this purpose.

1.4 Gaps in the State of the Art

Despite advancements in automated error detection tools and the initial success of large LLMs like GPT-4 in code generation, several critical gaps remain in the context of enhancing PDDL domain models for AI planning.

Limited Error Correction Capabilities. Existing tools, such as PDDL Studio and VAL, are adept at detecting syntactic and some semantic errors within PDDL domain files, but they are not designed to autonomously correct these issues. While these tools offer valuable insights into the nature of the errors, they typically require manual intervention by domain experts to make the necessary corrections. Particularly for more complex semantic errors—where logical inconsistencies arise between action preconditions and effects—there is currently no robust automated system in place that can refine domain models effectively [SDS+24]. This gap highlights the necessity for hybrid systems where error detection is automated but still allows for intelligent error correction mechanisms, potentially leveraging LLMs with deeper domain knowledge integration.

Lack of Domain-Specific Context Understanding. Although large language models such as GPT-4 can provide syntactically accurate PDDL domain models, they frequently lack semantic soundness. This restriction arises from the reality that LLMs, despite extensive training on substantial datasets, generally lack the specialized knowledge necessary to comprehend the complexities and limits of individual planning issues [KVG+24]. Consequently, LLMs may generate models that are logically incoherent or contravene fundamental limitations within the planning context. For example, they may produce actions with conflicting outcomes or neglect crucial predicates, resulting in erroneous plans. The research revealed that while LLM improvement significantly decreased syntactic mistakes, some intricate semantic issues remained unresolved. This underscores the necessity of including domain-specific information into LLMs to guarantee that the models generated conform to the logical framework of the domain.

Absence of a Measurable Improvement Process. A significant research gap exists in the development of standardized methods for assessing the quality of planning domain models, particularly regarding error reduction and plan optimization. Current research lacks established benchmarks or evaluation metrics for quantifying improvements in domain models generated or enhanced by LLMs, as well as more traditional approaches. As McCluskey et al. (2017) highlight, there is no unified notion of domain model quality, making it challenging to measure attributes like accuracy, completeness, and operability systematically. This absence of universally recognized benchmarks often results in qualitative or subjective evaluations, which may not fully capture practical performance in diverse planning applications [MVV17]. More comprehensive, standardized metrics are needed—beyond syntactic correctness—to evaluate model utility in real-world scenarios, ensuring that LLM-refined models can produce robust and efficient plans across varied domains.

1.5 Research Question

To address these gaps, this thesis explores the following research question: How can Large Language Models (LLMs) be utilized to improve planning domain models independently of human intervention, ensuring that the improved domain models (1) do not regress the properties of the original models and (2) improve the baseline domain models in a verifiable and measurable way?

The focus is on two critical criteria for the LLM-enhanced models:

No Regression in Properties. Any improvements made by the LLM must not lead to the degradation of the original model's functionalities. This is a significant challenge because, while LLMs can correct certain issues, they might inadvertently introduce new errors or alter essential components of the original model. Therefore, the enhanced model must maintain the core properties, constraints, and structures of the initial domain model. For instance, predicates and action effects in the original model must be preserved unless there is a valid reason to modify them.

Verifiable and Measurable Improvements. Improvements must not only be qualitative but also quantitatively verifiable. This is where evaluation metrics such as the domain model correctness and plan performance metrics come into play. The thesis seeks to establish whether LLMs can reduce both syntactic and semantic errors, and whether these improvements can be measured in a consistent, reliable way. The introduction of standardized, objective metrics allows the enhancement process to be rigorously evaluated, moving beyond subjective assessment to measurable outcomes.

1.6 Methodology

This thesis employs a system including many fundamental steps designed to utilize LLMs for the autonomous improvement of PDDL domain models, rectifying both syntactic and semantic inaccuracies. The principal phases for implementation are as follows:

1. **Design of the LLM-Based Enhancement Approach:** The next step is devising a methodology that utilizes LLMs to improve and augment current PDDL domain models.

The enhancing method based on LLM was formulated utilizing Chain-of-Thought (CoT) prompting [WWS+22].

- The procedure entails prompt-tuning the LLM using syntactic and semantic mistake instances derived from deficient domain models.
- Utilizing phased prompts to instruct the LLM in identifying and rectifying problems while generating enhanced domain models.
- Employing CoT prompting to improve logical reasoning for error rectification while preserving the characteristics of the original domain model.

2. **Development of Evaluation Metrics:** A major element of this thesis is the identification of criteria that can objectively assess the efficacy of LLM-based improvements to PDDL domain models. We want a metric that evaluates enhancements by quantifying the decrease of syntactic and semantic mistakes before to and following LLM refining. This measure would offer a dependable foundation for assessing mistake correction precision. Furthermore, to evaluate the quality of the plans produced by the updated models, measures such as makespan and plan cost are essential. These metrics will assess if enhancements to the domain models yield more efficient and optimum planning results.
3. **Experimental Validation:** We perform tests utilizing several International Planning Competition (IPC) benchmark models to assess the efficacy of LLM-enhanced domain models. Both imperfect and LLM-augmented domain models are submitted to PDDL Studio to evaluate error detection and repair functionalities. Plans are produced using the Temporal Fast Downward (TFD) planner [REM08] and verified with the VAL validator [SHS+22], emphasizing enhancements in error minimization and plan efficacy.
4. **Analysis of Results:** The concluding stage involves examining the experimental outcomes by comparing the pre- and post-refinement models for error reduction, plan quality, and computing efficiency.

1.7 Original Contribution

This thesis makes several original contributions to the field of AI planning and domain modeling:

1. The development of an LLM-based approach for refining and enhancing PDDL domain models.
2. The creation of the Enhance PDDL Correctness Score (EPCS), a new metric for evaluating the quality of domain models before and after LLM-based refinement.
3. An analysis of the potential and limitations of LLMs in domain model enhancement, highlighting key areas for future research.

1.8 Structure

The structure of this thesis is designed to guide the reader through the development, implementation, and evaluation of an LLM-based approach for improving PDDL domain models. The key chapters are organized as follows:

- **Chapter 2** provides a detailed background on AI planning and domain modeling, focusing on the Planning Domain Definition Language (PDDL) and its significance within planning systems. It explores the types of syntactic and semantic errors that can arise in domain models, laying the groundwork for subsequent discussions on enhancing these models.
- **Chapter 3** reviews the existing literature and state-of-the-art methodologies for PDDL model generation, validation, and error correction. A special emphasis is placed on the role of Large Language Models (LLMs) in automating these processes, addressing the capabilities and limitations of current tools and frameworks.

- **Chapter 4** outlines the design and implementation of the LLM-based approach for PDDL model refinement. It details the architecture and methodology behind the LLM-based system proposed in this thesis. It describes metrics such as Makespan, Plan Cost, and the EPCS, which were used to measure the effectiveness of the LLM-enhanced models.
- **Chapter 5** presents the experimental setup, including the benchmarks, evaluation process and metrics used to evaluate the effectiveness of the proposed approach. The integration of PDDL Studio and other relevant tools is also discussed in this chapter.
- **Chapter 6** discusses the results of the experiments, providing a detailed analysis of the impact of LLM-based refinement on model quality and planning efficiency. It provides insights into how the LLM-based approach improved (or failed to improve) the quality of domain models, focusing on error reduction and improvements in planning efficiency. The chapter also includes a discussion of trends observed across different domain models and the limitations of the approach.
- **Chapter 7** concludes the thesis with a summary of the key findings, a discussion of the limitations of the current approach, and suggestions for future research.

2 Background Information

2.1 AI Planning

AI planning is a crucial subfield of artificial intelligence that focuses on creating sequences of actions that allow systems to achieve specific goals. Whether it's a robotic system, a smart building, or a logistics operation, AI planning is used to automate decision-making by defining the steps required to reach an objective. The core challenge in AI planning lies in generating plans that are both efficient and executable, based on a predefined domain model [GNT04].

At the heart of AI planning is the need to represent knowledge about the environment and actions that the system can take [VHCM15]. This is done through domain models, which encode actions, objects, and the relationships between them. The most commonly used language for defining these models is the Planning Domain Definition Language (PDDL). PDDL provides a formal framework that allows planners to generate, simulate, and optimize sequences of actions, ensuring that the system can move from an initial state to a goal state while satisfying various constraints [SHS+22].

2.2 Planning Domain Definition Language

PDDL is a high-level, domain-specific language used in AI planning to define both domain models and problem instances. Introduced in the late 1990s, PDDL was designed to provide a standardized way of representing the actions and predicates of a planning problem, making it easier to apply automated planners across different domains [HLM+19].

A PDDL domain file defines:

- **Types:** The categories of objects involved (e.g., blocks, locations, robots).
- **Predicates:** Descriptions of the relationships between objects (e.g., 'on', 'holding').
- **Actions:** Descriptions of possible operations, which include preconditions (what must be true for the action to execute) and effects (the result of the action).

2 Background Information

Listing 2.1 PDDL Domain for Blocksworld

```
(define (domain blocksworld)
  (:requirements :strips :typing)
  (:predicates
    (on ?x ?y)
    (ontable ?x)
    (clear ?x)
    (handempty)
    (holding ?x)
  )
  (:action pick-up
    :parameters (?x)
    :precondition (and (clear ?x) (ontable ?x) (handempty))
    :effect (and (holding ?x) (not (clear ?x)) (not (ontable ?x)) (not (handempty))))
  )
  (:action put-down
    :parameters (?x)
    :precondition (holding ?x)
    :effect (and (clear ?x) (ontable ?x) (handempty) (not (holding ?x))))
  )
  ;; Additional actions for stacking and unstacking would go here
)
```

As shown in Listing 2.1, the Blocksworld domain ¹ defines predicates such as ‘on’, ‘ontable’, and actions like ‘pick-up’ and ‘put-down’.

A PDDL problem file contains:

- **Objects:** Specific instances of types in the domain.
- **Initial state:** The starting configuration of the world.
- **Goal state:** The desired configuration the system must achieve.

¹<https://github.com/jan-dolejsi/vscode-pddl-samples/blob/master/Blocksworld/blocksworld.pddl>

Listing 2.2 PDDL Problem for Blocksworld

```
(define (problem blocksworld-problem)
  (:domain blocksworld)
  (:objects
    A B C - block
  )
  (:init
    (ontable A) (ontable B) (ontable C)
    (clear A) (clear B) (clear C)
    (handempty)
  )
  (:goal
    (and (on A B) (on B C))
  )
)
```

As shown in Listing 2.2, the problem file for Blocksworld defines the initial state with blocks ‘A’, ‘B’, and ‘C’ on the table and the goal state as stacking ‘A’ on ‘B’ and ‘B’ on ‘C’.

2.3 Common Errors in PDDL Domain Files

PDDL models must be both syntactically and semantically correct for planners to generate valid plans [SB23]. Errors in PDDL files can lead to issues such as incomplete plans, invalid actions, or planners failing to generate any plan at all. These errors fall into two main categories: syntactic errors and semantic errors.

2.3.1 Syntactic Errors

Syntactic errors arise from incorrect formatting or structure in the PDDL file. Common syntactic errors include:

- **Missing parentheses:** Since PDDL is a Lisp-like language, parentheses are critical for structuring predicates and actions. A missing or misplaced parenthesis can make the entire file unreadable by the planner.
- **Incorrect keyword usage:** Misusing PDDL keywords (like: precondition or: effect) can result in invalid action definitions.
- **Undeclared objects or actions:** Failing to declare objects or types in the domain file can cause issues when those objects are referenced in the problem file.

2.3.2 Semantic Errors

Semantic errors occur when the PDDL model is logically inconsistent, even though the syntax is correct ². These errors can lead to plans that are impossible to execute. Common semantic errors include:

- **Unsatisfied preconditions:** An action is defined with preconditions that can never be satisfied in the given domain. For example, a robot might be required to be in two places at once, making the action logically impossible.
- **Redundant effects:** These occur when the effects of an action don't change the state because the action's preconditions already fulfill those effects. For instance, an action might include an effect that a block is on a table when it was already on the table.
- **Complementary effects:** This happens when an action has contradictory effects, such as setting a predicate to both true and false simultaneously.

2.4 Why Errors in PDDL Matter

Errors in PDDL files can significantly impact both the performance of AI planners and the quality of the plans they generate. A single syntactic mistake, such as a missing parenthesis or incorrectly defined predicate, may prevent the planner from interpreting the domain file, leading to a failure in generating any plan [PCBB12]. More importantly, semantic errors, which involve logical inconsistencies in the domain model, can cause the planner to produce suboptimal plans or none at all. These errors may arise from actions with unsatisfied preconditions, contradictory effects, or missing dependencies, which could lead to failures in plan execution or unnecessary actions.

A semantic mistake, such as inadequately defining preconditions for an action, may lead the planner to pursue illogical activities, so squandering computational resources and time. Likewise, conflicting outcomes inside activities might hinder the planner from attaining the intended objective, even with appropriate initial circumstances. Consequently, rectifying faults in PDDL files is essential for guaranteeing the reliability and effectiveness of AI planning systems. In the absence of adequate validation and corrective processes, these mistakes may compromise the efficacy of automated planning systems, making them inefficient in practical applications.

2.5 Error Detection and Evaluation Tools

To assist with error detection and refinement in PDDL domain files, several tools have been developed:

- **PDDL Studio:** This tool provides a visual interface for creating and validating PDDL domain and problem files. It checks for common syntactic and semantic errors and helps users ensure their models are valid before submitting them to a planner [PCBB12].

²[semantic-error-chapter-80-understanding-detecting](#)

- **VAL (Validator for PDDL):** VAL is a widely-used tool for verifying the validity of PDDL domain models. It checks whether the generated plans are executable within the specified domain and identifies any inconsistencies in the model [SHS+22].

2.6 Large Language Model (LLM) and Prompting

LLMs, like GPT-3 and BERT, are deep neural networks engineered to comprehend and produce human language. These models, constructed from extensive datasets, utilize billions of parameters to forecast and generate text in response to input prompts, showcasing a formidable capacity to comprehend language patterns, semantics, and context [ZZL+23].

CoT prompting is a method in natural language processing that directs a LLM to address intricate inquiries by systematically deconstructing reasoning into intermediary steps. This methodology enables models to more effectively manage tasks necessitating sequential reasoning by promoting the processing of one logical step at a time, hence enhancing interpretability and accuracy (Wei et al., 2022). Utilizing CoT, LLMs can attain more dependable results in tasks requiring reasoning, such as mathematical word problems or multi-step decision-making [WWS+22].

3 Related Work

The article 'Issues in Planning Domain Model Engineering' by McCluskey, Vaquero, and Vallati discusses the difficulties in developing domain models for automated planning systems, which function as formal definitions of domains. It underscores the necessity for exact definitions and characteristics of domain models to address engineering challenges efficiently. The project aims to improve the comprehension and techniques related to the development, validation, and optimization of domain models in the AI planning community. Contributions encompass the proposal of a series of definitions and criteria for evaluating domain model quality, while underscoring the significance of model completeness, operability, and efficiency. This research is pertinent to our thesis subject as it offers essential insights and methodologies for enhancing the quality and efficacy of domain models, which may be further augmented through the utilization of LLMs [MVV16].

The research article 'Software Development Life Cycle for Engineering AI Planning Systems' introduces a software development life cycle named PlanXflow, comprising 10 phases for the design, development, deployment, and utilization of planning systems. This study aims to examine the intricate process of constructing and overseeing AI planning systems, emphasizing the necessity of developing appropriate planning models, choosing effective planning tools, implementing production deployment, and recognizing sometimes overlooked tasks. This study contributes by presenting a systematic methodology for designing AI planning systems, addressing unresolved research and development difficulties, and establishing a foundation for discourse in the domains of AI planning and AI engineering. This study pertains to our subject as it underscores the significance of building and verifying planning domain models, which corresponds with the improvement of domain models for AI planning that we are concentrating on in our thesis [Geo23].

The research paper 'Impact of Large Language Models on Generating Software Specifications' explores the performance of Large Language Models (LLMs) in generating software specifications, focusing on the comparison of LLMs with traditional methods and the impact of prompt construction strategies. The purpose of the study is to evaluate the effectiveness of LLMs in software specification generation and to identify their unique strengths and weaknesses. The contributions of the paper include insights into the performance of LLMs, the evaluation of prompt construction strategies, and a comprehensive failure diagnosis. This work is relevant to our thesis topic as it provides valuable insights into the capabilities and limitations of LLMs, which can be applied to enhance domain models for AI planning, particularly in the context of software specification generation [XYJ+23].

The research article 'LLM-assisted Knowledge Graph Engineering: Experiments with ChatGPT' investigates the capabilities of LLM, specifically ChatGPT, in facilitating knowledge graph engineering activities. It investigates studies assessing the proficiency of LLMs in constructing and modifying knowledge graphs, semantic search, and SPARQL query formulation. The objective of the research is to examine how ChatGPT might facilitate knowledge graph engineering and illustrate its possible applications within this field. The work's contributions encompass demonstrating the viability of AI-driven query creation, underscoring the necessity for result validation, and accentuating the significance of open training corpora

3 Related Work

and large language models for scientific inquiry. This research is pertinent to our thesis as it offers insights into the application of LLMs for the augmentation of domain models, specifically within the realm of knowledge graph engineering, which corresponds with our emphasis on LLM-driven improvement of domain models for AI planning [MSF+23].

The research paper 'Abstractions, Scenarios, and Prompt Definitions for Process Mining with Large Language Models: A Case Study' illustrates the capabilities of LLMs in process mining by abstracting process mining artifacts and crafting prompts for these models to address process-related inquiries. It seeks to demonstrate the use of LLMs, including GPT-4, in comprehending and analyzing intricate process architectures, presenting diverse prompting tactics to mitigate information loss from abstractions. The contributions encompass the proposal of abstractions for typical process mining artifacts, the presentation of prompting tactics, and the execution of a case study utilizing publicly accessible event logs. This study is pertinent to our subject as it corresponds with the improvement of domain models for AI planning through the utilization of natural language comprehension and process interpretation skills of LLMs [BSA23].

The research article 'Leveraging Pre-trained Large Language Models to Construct and Utilize World Models for Model-based Task Planning' examines the application of pre-trained LLMs in the construction and utilization of world models for model-based task planning. The objective of this study is to overcome the constraints of using LLMs directly as planners by proposing an innovative paradigm that formulates explicit world (domain) models in PDDL and subsequently utilizes them for planning with reliable domain-independent planners. This study demonstrates the possibility of utilizing LLMs to rectify PDDL models based on feedback and effectively translates PDDL into normal language for users without a PDDL background. This research is pertinent to our thesis as it offers insights on utilizing LLMs to enhance the accuracy and utility of domain models for planning tasks, consistent with the objective of improving domain models for AI planning through LLMs [GVSK23].

The research paper 'On the Prospects of Incorporating Large Language Models (LLMs) in Automated Planning and Scheduling (APS)' examines the integration of LLMs with AI planning, emphasizing reasoning, task planning, and language comprehension. The objective of the research is to examine the potential of LLMs in improving domain models for AI planning by utilizing their language production and comprehension skills. This study proposes methodologies like fine-tuning and in-context learning to tailor pre-trained LLMs for specific task domains, and investigates the application of LLMs in embodied task planning and open-world multi-task agents. The work is directly pertinent to our topic as it offers insights into the application of LLMs for reasoning, task planning, and language comprehension, thus providing valuable knowledge and methodologies for improving domain models within the realm of AI planning [PMR+24].

The study by Vallati et al. (2015), entitled 'On the Effective Configuration of Planning Domain Models,' investigates the significance of optimizing PDDL domain model configurations to enhance the efficacy of domain-independent planners such as Fast Downward and LAMA. The authors suggest an automated approach for configuring domain models by modifying the sequence of predicates, operators, and conditions, demonstrating that minor adjustments in model configurations can result in significant enhancements in plan generation time. Their findings demonstrate that this technique may produce substantial performance enhancements across several planners and domains, achieving speedups of up to 300 times in certain instances. Nevertheless, the methodology is constrained by its reliance on

planner-specific setups and may not be broadly applicable across all planning contexts. Our study advances this by emphasizing error correction and refinement, rather than mere configuration, to guarantee both syntactic and semantic accuracy of domain models utilizing LLMs such as GPT-4 [VHCM15].

The paper 'On the Planning Abilities of Large Language Models: A Critical Investigation' by Valmeekam et al. critically examines the planning capabilities of LLMs, particularly GPT-4, in autonomous plan generation and as heuristic guides for AI planners. Their findings indicate that while LLMs struggle with autonomous planning, achieving only a 12% success rate in generating executable plans, they show more promise when used to assist AI planners by providing heuristic guidance. In this mode, LLM-generated plans can help external planners like LPG correct and improve plans. This aligns with our thesis, where we leverage LLMs, specifically GPT-4, to generate efficient and error-free PDDL domain files using techniques like few-shot and chain-of-thought prompting. By evaluating the reduction of syntactic and semantic errors with tools like PDDL Studio, we aim to show how LLMs can enhance domain models for AI planning, complementing Valmeekam et al.'s findings on the potential of LLMs in planning tasks [VSM+23].

The paper 'Generalized Planning in PDDL Domains with Pretrained Large Language Models' by Silver et al. explores how GPT-4 can be used for generalized planning by synthesizing Python programs that generate valid plans for PDDL domains. The study builds on previous efforts in AI planning, particularly leveraging Chain-of-Thought (CoT) prompting and automated debugging to improve the model's planning performance. Compared to earlier approaches that often relied on domain-specific planners or search-based strategies, GPT-4's ability to generalize across tasks with minimal training is a key contribution. However, limitations include inconsistent performance across complex domains and reliance on PDDL names for effective planning, indicating that while GPT-4 shows promise, its generalization capabilities are still dependent on structured inputs and specific domain characteristics [SDS+24].

The article 'Generating Consistent PDDL Domains with Large Language Models' by Smirnov et al. investigates the application of LLMs to produce PDDL domains from natural language descriptions. The authors provide an innovative system that combines automated consistency verification and error rectification to tackle the difficulties of producing executable PDDL domains, therefore minimizing human involvement. This study advances prior research on LLMs that generate code or PDDL fragments by incorporating a feedback loop to rectify mistakes and guarantee goal attainability. Nonetheless, constraints persist, as semantic accuracy necessitates human verification, and the methodology has difficulties in increasingly intricate areas. The authors illustrate that whereas LLMs may significantly decrease human effort in producing PDDL domains, more enhancements are necessary for managing bigger, more intricate activities [SJCG24].

The article 'LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks' by Kambhampati et al. contends that Large Language Models (LLMs), including GPT-4, lack the ability for autonomous planning or self-verification. The authors contend that LLMs largely serve as approximate information sources rather than genuine planners, lacking the requisite reasoning capabilities for intricate planning tasks. The authors offer the LLM-Modulo framework, wherein LLMs create ideas or candidate plans that are then confirmed by external verifiers. This framework enhances prior initiatives utilizing LLMs in planning by including a more cohesive neuro-symbolic methodology, facilitating a closer connection between LLM-generated plans and model-based evaluators to guarantee soundness and accuracy. This

3 Related Work

method enhances plan accuracy; yet, the article underscores the constraints of depending on LLMs for complete autonomous reasoning and planning, necessitating external validation to rectify intrinsic deficiencies in LLM-generated plans [KVG+24].

The paper 'Fixing PDDL Domains Using Delta Debugging' by Eckstein et al. introduces an automated approach for identifying and fixing faulty preconditions in PDDL domain files using the Delta Debugging algorithm. This approach builds on previous work in planning validation tools, such as the Plan Validator, by focusing on isolating precondition errors without requiring prior knowledge of the system being debugged. The paper addresses a significant challenge in AI planning: the lack of feedback when domain errors occur. The authors demonstrate the debugger's effectiveness in identifying precondition errors across various domains from the International Planning Competition. However, the system is currently limited to preconditions and does not address errors in action effects or provide solutions for invalid plans generated due to overly relaxed constraints. This work improves upon brute-force debugging methods by significantly reducing the effort required to fix domain files, though further enhancements are needed to handle more complex errors in PDDL domains [EDS23].

The study entitled 'Large Language Models as Planning Domain Generators' (Oswald et al., 2024) examines the capability of LLMs to automate the creation of PDDLs domain models from natural language descriptions. This study largely emphasizes the automation of the historically laborious operation of domain model generation, a crucial element of AI planning that frequently necessitates domain expertise. The study specifically assesses the performance of seven LLMs across many domains, evaluating their capacity to produce accurate and comprehensive domain models. The research introduces an automated evaluation framework to overcome the shortcomings of subjective, expert-based evaluation techniques prevalent in previous studies, including those by Lindsay et al. (2017) and Guan et al. (2023) [GVSK23]. This method, although promising, reveals limits in the capacity of LLMs for profound semantic comprehension, suggesting that more refinement is necessary to develop high-quality, planner-compatible models [OSK+24].

4 Proposed Framework

The architecture for implementing the solution to enhance PDDL domain models using LLM consists of three key stages: Training Stage, Testing Stage and Evaluation Stage. Each phase involves distinct workflows that integrate various datasets, software tools, and planners, with the goal of refining domain models for AI planning. The architecture ensures that errors in PDDL domain models—whether syntactic or semantic—are identified, corrected, and validated to improve the overall performance of AI planners.

4.1 Training Stage

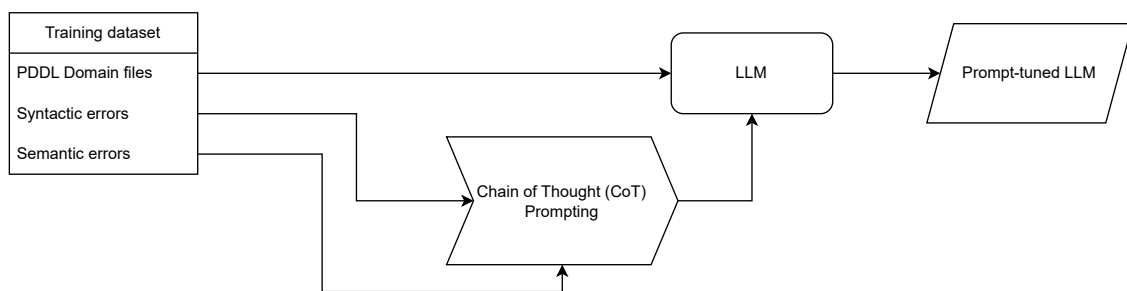


Figure 4.1: Training Architecture

The Training Stage is designed to instruct the LLM how to detect, analyze, and correct errors in PDDL domain models. This stage is crucial for shaping the capabilities of the LLM and ensuring it can effectively refine PDDL models with minimal human intervention. The primary goal is to prepare the LLM for the Testing Stage, where it will be tasked with enhancing new, unseen PDDL models. In Figure 4.1, the workflow for training a LLM to become 'Prompt-tuned LLM' for PDDL model enhancement is visualized. Here's a step-by-step explanation of the process illustrated in the diagram:

4.1.1 Training Dataset

The process begins with a carefully curated dataset consisting of PDDL domain models embedded with various types of semantic and syntactic errors. Semantic errors typically include issues related to logical inconsistencies, such as unsatisfied preconditions or contradictory effects. Syntactic errors, on the other hand, are structural mistakes, such as missing parentheses or incorrect keywords. This dataset acts as a foundational reference, enabling the LLM to learn from a diverse array of domain model errors commonly encountered in real-world planning problems.

4.1.2 Training Prompt Phases

To enable the LLM to effectively recognize, analyze, and correct these errors, a CoT prompting technique is employed. CoT prompting allows the LLM to break down complex tasks into logical steps, making the error detection and correction process more interpretable [LLLJ23]. The journey towards achieving a prompt-tuned LLM consists of two structured phases P1 and P2. Each phase is carefully crafted to enhance the LLM's understanding and detection of errors in PDDL domain files.

Phase 1 (P1): Making the LLM Aware of Various PDDL Domain Errors

Prompt P1 is structured into three main components to guide the LLM in understanding the foundational errors in PDDL domain files:

1. **Introduction:** The prompt begins by introducing the LLM's role, emphasizing its responsibility to identify and correct errors in PDDL files. This section establishes the context, framing the task as critical for enhancing AI planning effectiveness.
2. **Error Categories:** Next, the prompt categorizes errors into two primary types: Syntactic Errors and Semantic Errors. This high-level categorization helps the LLM differentiate between structural issues and logical inconsistencies within PDDL models.
3. **Focus and Objective:** The final component of the prompt clarifies the objective of Phase 1, instructing the LLM to focus on understanding how each error type impacts the overall validity of the domain file. It signals that detailed examples will follow in the next phase, ensuring a gradual learning progression.

Phase 2 (P2): Providing Examples of Flawed PDDL Domain Files

Prompt P2 follows a structured approach similar to P1, but with added depth to guide the LLM in identifying and addressing specific error types through examples. The breakdown of P2 includes:

1. **Objective:** It establishes that the LLM will examine flawed PDDL files with explanations of each error, helping it learn error correction techniques.
2. **Error examples:** The LLM is provided with examples of syntactic and semantic errors, each accompanied by a code snippet.
3. **Error Explanation:** For each code snippet, the prompt includes a brief explanation detailing why the error affects the PDDL file. This explanation reinforces the error type and its impact.
4. **Correction advice:** The LLM is prompted with correction to the aforementioned errors.

4.1.3 LLM Processing

Throughout the training phase, the LLM undergoes iterative learning, where the outputs generated by the model are fed back into the training process. This feedback loop allows the LLM to learn from its mistakes and progressively improve its ability to correct PDDL models. The process where the prompts used to guide the LLM are refined over multiple iterations is called “Prompt-Tuning”. Initially, simpler prompts may be used to correct basic syntactic errors. As the LLM becomes more adept, more complex prompts are introduced to address semantic issues and domain-specific challenges. This iterative process ensures that the LLM is fine-tuned to handle a wide variety of PDDL domains and error types [GKS24]. As the LLM interacts with the CoT prompts, it learns to apply structured reasoning to analyze and resolve issues within PDDL files. Through repeated exposure to error-laden examples and corrective reasoning, the model adapts to become more proficient in error detection, particularly in identifying and differentiating between syntactic and semantic issues.

4.1.4 Prompt-tuned LLM

The final output of this training process is a Prompt-tuned LLM. This enhanced model is capable of examining PDDL domain files and effectively detecting and correcting both semantic and syntactic errors. The result is an LLM that can autonomously validate and refine PDDL models, thereby improving the reliability and robustness of domain models used in AI planning.

4.2 Testing Stage

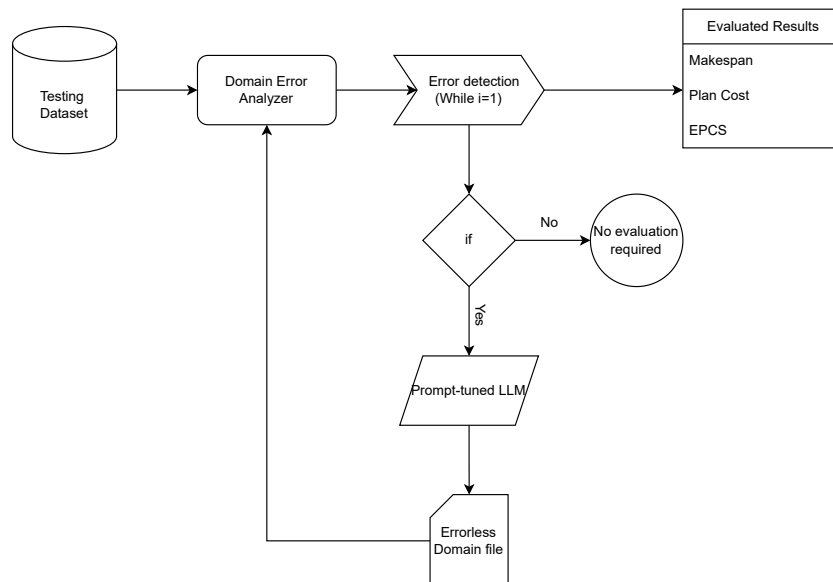


Figure 4.2: Testing Architecture

The Testing Stage assesses the efficacy of the LLM trained in the preceding phase. At this step, the LLM encounters a novel testing dataset of PDDL domain models that it has not before seen. The testing step is essential for evaluating the generalization skills of the LLM and confirming its capacity to rectify mistakes in unfamiliar contexts. Figure 4.2 depicts the evaluation of a Prompt-tuned LLM, specifically optimized for identifying and rectifying faults in PDDL domain files, utilizing a specialized Testing Dataset.

4.2.1 Testing Dataset

The testing dataset covering a wide range of planning problems. These domains include:

- **Well-formed models:** Some of the models are syntactically and semantically correct. These are used to evaluate whether the LLM can recognize a correctly formed model and avoid making unnecessary changes.
- **Flawed models:** These contain the same types of errors (syntactic and semantic) present in the training dataset, but the specific errors and domains are new to the LLM. This allows us to test how well the LLM generalizes its error-correction capabilities.

The purpose of using this dataset is to assess how well the LLM performs when confronted with new challenges. By testing the LLM on unseen domains, we can ensure that it is not simply memorizing error patterns from the training data but has learned to reason through PDDL models independently.

4.2.2 Domain Error Analyzer

The procedure starts with a testing dataset of PDDL domain files, which are first subjected to a Domain Error Analyzer. This analyzer identifies any existing syntactic or semantic flaws in the models. Should no mistakes be identified, the procedure terminates with 'No Evaluation Required'. It does an exhaustive syntactic and semantic analysis of the PDDL domain models. The syntactic analysis verifies structural accuracy, ensuring the model complies with PDDL's formal language, including appropriately defined predicates, actions, and accurate parenthetical use. Simultaneously, semantic analysis recognizes more intricate problems, such as unmet preconditions, conflicting consequences, or superfluous acts, which may compromise the logical coherence of the planning domain. The testing step encompasses an error detection loop (While $i=1$), during which the Domain Error Analyzer persistently examines each PDDL domain file for problems. This loop guarantees comprehensive assessment of all domain files for syntactic and semantic discrepancies.

4.2.3 LLM Enhancement of PDDL Models

If the Domain Error Analyzer identifies errors, the process continues with the intervention of the Prompt-tuned LLM, which is specifically trained to address and correct these detected issues. In this phase, the prompt-tuned LLM is tasked with enhancing the PDDL domain models by identifying and correcting errors. The workflow involves feeding the testing dataset into the Prompt-tuned LLM and generating enhanced versions of the domain models. Here, additional prompt phases viz. P3 and P4 are utilized to guide the LLM in generating improved, error-free PDDL domain models. The prompt phases P3 and P4 are as follows:

Phase 3 (P3): Testing with a Benchmark PDDL Domain File and Error Identification

Prompt P3 has a concise structure designed to guide the LLM in analyzing an error-prone PDDL domain file. Here's the breakdown:

1. **Objective:** Test the LLM's ability to identify errors in a new PDDL domain file based on previous training.
2. **Benchmark PDDL Domain File:** The LLM is presented with testing domain file, containing a set of predicates and actions that describe a simple environment.
3. **Task Instructions:** The LLM is instructed to identify any syntactic and semantic errors present in the domain file.
4. **Error Detection:** Providing a list of detected issues, accompanied by explanations for each identified problem.

Phase 4 (P4): Generating an Errorless PDDL Domain File for Efficient Planning

Prompt P4 guides the LLM in creating an error-free and efficient PDDL domain model. Here's the structure:

1. **Objective:** Use the knowledge gained to generate a correct and optimized PDDL domain file that is free of errors and efficient for planning.
2. **Domain Constraints:** Specifies the domain model requirements, including logical definitions for predicates, actions, preconditions, and effects, ensuring clarity and coherence in the domain structure.
3. **Output Expectations:** Reinforces that the output must be optimized for effective planning, showcasing an understanding of domain organization and error prevention.

The enhanced PDDL domain files produced by the LLM are then subjected to further validation using Domain Error Analyzer, which checks for any remaining syntactic or semantic errors. This ensures that the enhanced models are robust and error-free.

This process draws on the techniques described in literature such as [GKS24] and [CKD24], where structured prompting and iterative feedback loops improve the performance of models in complex, domain-specific tasks. By using the prompts outlined in P3 and P4, the LLM is transformed into a prompt-tuned model that can autonomously enhance PDDL domain files, providing a significant improvement in error correction and domain consistency.

4.3 Evaluation Stage

The final output of the testing phase is an Evaluated project file containing the pre-LLM influenced and enhanced domain models. These models are evaluated based on the quality of the plans generated from them, as well as the number of errors corrected. Metrics which judges error reduction and plan efficiency are used to quantify the improvements achieved by the Prompt-tuned LLM.

This phase serves as a comprehensive test of the LLM’s capabilities, ensuring that it can generalize its error-correction skills across multiple domains and produce high-quality, valid PDDL models that lead to efficient AI plans.

4.3.1 Enhance PDDL Correctness Score (EPCS)

The EPCS metric is a specialized assessment measure created to evaluate the efficacy of enhancements implemented in PDDL domain files, particularly on the reduction of syntactic and semantic mistakes. The main goal of EPCS is to measure the advancements attained by utilizing LLMs in rectifying and refining the domain models. The EPCS formula draws from several metrics that assess mistake reductions and temporal improvements, applying them to PDDL domain models with a particular emphasis on syntactic and semantic faults.

The EPCS score reflects the relative reduction in errors between the initial, error-prone domain file (pre-LLM enhancement) and the final, LLM-enhanced domain file. It compares the total number of errors, both syntactic and semantic, before and after the LLM has been applied to the domain model. The EPCS score is calculated using the following formula:

$$(4.1) \text{ EPCS} = 100 \times \left(1 - \frac{S_e + M_e}{S_e^{\text{pre}} + M_e^{\text{pre}}} \right)$$

where:

- S_e : Number of syntactic errors after LLM enhancement
- M_e : Number of semantic errors after LLM enhancement
- S_e^{pre} : Initial number of syntactic errors before enhancement
- M_e^{pre} : Initial number of semantic errors before enhancement

The Equation (4.1) was inspired by evaluation metrics commonly found in error-reduction or performance improvement studies [HLM+19] in AI-related tasks. The formula is structured to reflect the relative improvement in the number of semantic and syntactic errors before and after the LLM-based enhancement. This method of calculating error reduction is loosely based on ideas from relative error reduction and improvement score calculations [BBF+14], which are common in various fields where performance evaluation is based on reductions in errors, defects, or inefficiencies [EBSD21].

4.3.2 Key performance Metrics

Once the errors in the enhanced domain models have been corrected or minimized, the models are passed to AI planners, to generate action plans. The planner processes the enhanced PDDL models and generates a sequence of actions that achieve the specified goal state from the initial state.

The quality of the generated plans is then evaluated using key performance metrics:

Makespan

Makespan is the time difference between the start of the first action and the end of the last action in the plan, assuming time-based (temporal) actions. This metric is critical for evaluating the efficiency of a generated plan, as a lower makespan indicates a more efficient plan that completes tasks in less time [KSV+13]. Makespan is particularly useful in domains where time is a limiting factor, such as logistics, manufacturing, and resource management [HT23].

Here's a general formula for makespan:

$$(4.2) \text{ Makespan} = \max(\text{end_time}(a_1), \text{end_time}(a_2), \dots, \text{end_time}(a_n))$$

Where:

- a_1, a_2, \dots, a_n are the actions in the plan.
- $\text{end_time}(a_i)$ represents the end time of action.

In sequential plans (where actions happen one after another with no overlap), the formula can be written as:

$$(4.3) \text{ Makespan} = \sum_{i=1}^n \text{duration}(a_i)$$

Where:

- $\text{duration}(a_i)$ is the Duration of action a_i .

Plan Cost

Plan Cost in general refers to the cumulative 'cost' of executing a plan, which could encompass various factors, such as the number of actions in the plan, the resource consumption of each action, or specific costs assigned to certain actions. Unlike makespan, which focuses solely on time, plan cost provides a broader perspective by considering the resources or effort required to carry out the plan [RMT14]. Plan cost is an important metric for evaluating the efficiency of a plan in terms of resource utilization. A lower plan cost indicates that the plan is resource-efficient, while a higher plan cost might indicate redundancies or an overuse of resources, which could be detrimental in scenarios where resources are limited [KSV+13].

Why Makespan and Plan Cost?

The combination of Makespan and Plan Cost provides a comprehensive evaluation of a plan's efficiency, covering both time and resource utilization. In many AI planning applications, it's not enough to generate a plan that achieves the goal; it is equally important to ensure that the plan does so in a cost-effective and timely manner. By using these two metrics, we can assess whether the LLM-generated plans are not only correct but also optimized for real-world execution, where time and resources are critical factors.

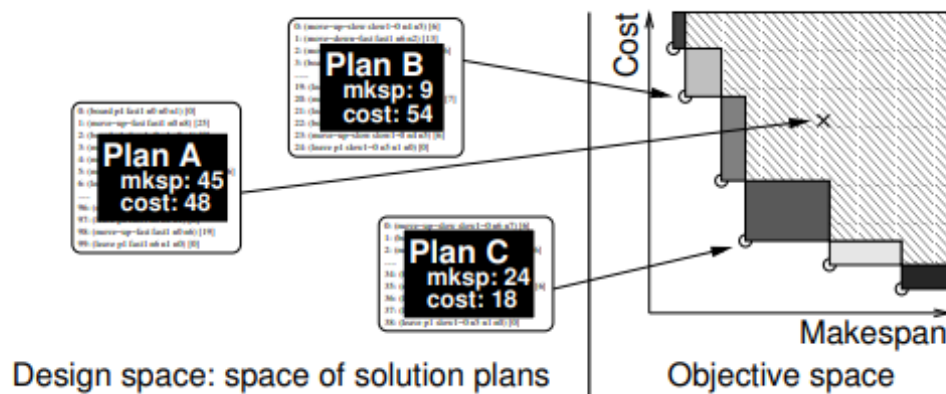


Figure 4.3: Design and objective spaces for a two-objectives planning problem [KSV+13]

The Figure 4.3 shows how plans exist within a Pareto front, where certain solutions dominate others by being superior in terms of both cost and makespan (33). These evaluations enable planners to identify the best possible solution that minimizes both time and resources, making these metrics invaluable when comparing different planning approaches.

5 Implementation

The effective execution of the LLM-based augmentation of PDDL domain models depends on the smooth integration of various components throughout the Training and Testing Stage. This chapter delineates the distinct phases of development, encompassing data preparation, model training, and testing, along with the selection of tools and methodologies employed to attain sophisticated domain models. Every element and method in this chapter adds to a unified workflow designed to reduce human involvement and improve the resilience of PDDL models for practical planning tasks.

5.1 Training Dataset

The training dataset utilized to educate the LLM on mistake detection and rectification is sourced from a collection of defective planning models, accessible here¹. This dataset is meticulously vetted to include PDDL models that exhibit intentional syntactic and semantic faults. The multitude and diversity of mistakes present in this dataset provide it an exceptional resource for training the LLM to identify prevalent flaws across PDDL domain models.

Relevant Features of the Training Dataset. The dataset comprises PDDL domain models with various mistakes, including unfulfilled preconditions, duplicated effects, absent parameters, and syntax difficulties such as missing parentheses. These prevalent faults assist in training the LLM to identify and suggest rectifications for many categories of problems.

Domain Variety. The dataset encompasses many planning domains, including logistics, block world, and robotic motion tasks. This guarantees that the LLM encounters several planning frameworks, allowing it to generalize effectively across diverse domains when suggesting improvements to PDDL files.

Error Frequency and Distribution. The dataset is intentionally defective, resulting in a balanced error distribution that offers several instances of both grammatical and semantic problems. This facilitates a more rigorous training process, enabling the LLM to prioritize and rectify significant faults while preserving the logical coherence of the domain models.

5.2 Choice of LLM

Based on evidence, ChatGPT-4 (GPT-4) has been selected as the primary Large Language Model (LLM) for enhancing PDDL domain models. Compared to other LLM's, GPT-4 introduces significant improvements in comprehension, reasoning, and error-correction capabilities—making it particularly well-suited for the task of refining complex AI planning models.

¹[flawedPlanningModels](#)

5.2.1 Why GPT-4?

GPT-4 is a substantial upgrade from its predecessor, GPT-3. It features improved performance across several dimensions compared to other LLM's that are highly relevant for enhancing domain models:

- **Larger Model and Contextual Understanding:** GPT-4 has more parameters and a larger context window than any other LLM, allowing it to handle more complex inputs and generate coherent, nuanced outputs even when dealing with long or intricate planning domains ².
- **Advanced Reasoning Abilities:** GPT-4 has shown superior performance in reasoning tasks, including multi-step problem solving and more complex logical deductions, which are essential for detecting and correcting both syntactic and semantic errors in PDDL domain models ³.
- **Improved Accuracy:** GPT-4 reduces 'hallucinations' (where models generate incorrect information) and offers a more precise understanding of prompts, leading to fewer errors when processing highly structured text like PDDL ⁴.

The Table 5.1 ^{5 6} below provides a comparison of different GPT models, highlighting GPT-4's advantages in handling complex, structured tasks like PDDL domain model enhancement:

Feature	GPT-3	GPT-4	LLaMa-70B
Parameters	175 billion	100+ trillion	65 billion
Context Window	2048 tokens	4096 tokens	4096 tokens
Error Handling	Moderate	Advanced	Moderate
Reasoning Skills	Basic	Advanced	Moderate
Syntactic Error Detection	Good	Excellent	Good
Semantic Error Detection	Limited	Excellent	Moderate
Cost Efficiency	Lower	Higher	Moderate
Plan Generation Precision	Moderate	Excellent	Moderate

Table 5.1: Comparison of Features between GPT-3, GPT-4, and LLaMa

To enhance the accuracy and robustness of PDDL domain models, this approach uses GPT-4 with CoT prompting techniques. As discussed before in Section 4.1.2, through a structured series of prompts P1 and P2 in the training phase, GPT-4 learns to detect and correct both syntactic and semantic errors in domain models. The result of this iterative prompting and feedback process is a 'Prompt-Tuned' LLM, specifically adapted for improving PDDL models. This refined model is then capable of identifying errors effectively and generating corrected, optimized versions of PDDL files, reducing the need for extensive manual intervention.

²[//www.vectara.com/](https://www.vectara.com/)

³[//www.aimunch.com/](https://www.aimunch.com/)

⁴[//www.33rdsquare.com/](https://www.33rdsquare.com/)

⁵[//www.toolify.ai/](https://www.toolify.ai/)

⁶[//www.techrxiv.org/](https://www.techrxiv.org/)

5.2.2 Prompt-Tuning LLM

GPT-4 is exposed to the fundamental types of errors found in PDDL domain models, specifically focusing on Syntactic and Semantic errors. The goal of P1 is to ensure that the LLM comprehends the types of flaws that can occur in PDDL files and how they disrupt both syntax and logic.

P1 Example

LLM Prompt
<p>You are an AI model trained to specialize in detecting and correcting errors in PDDL (Planning Domain Definition Language) domain files, which are critical for successful AI planning. In this phase, your task is to learn how to identify two types of common errors in PDDL files: Syntactic Errors and Semantic Errors.</p> <p>1. Syntactic Errors: These occur when the structure of the PDDL file is incorrect.</p> <ul style="list-style-type: none"> • Missing or mismatched parentheses. • Incorrect keyword usage or undefined predicates. • Improper formatting of action definitions or predicates. <p>2. Semantic Errors: These errors involve logical inconsistencies within the PDDL domain model, affecting the planning logic.</p> <ul style="list-style-type: none"> • Action preconditions contradicting their effects. • Undefined or improperly used predicates in action preconditions or effects. • Redundant or missing actions causing incomplete or overly complex plans. <p>In this phase, focus on learning how these errors manifest and how they affect the correctness of the domain file. For each error type, I will provide detailed examples in the next phase.</p>
LLM Output
<p>I'll focus on understanding how Syntactic and Semantic Errors impact PDDL files for AI planning. I'll review the structural aspects to spot issues like missing or mismatched parentheses, incorrect keywords, and formatting problems for Syntactic Errors. For Semantic Errors, I'll concentrate on identifying logical inconsistencies, like contradictory action preconditions and effects, undefined predicates, and issues with action completeness and complexity. Looking forward to the detailed examples in the next phase!</p>

P2 Example

LLM Prompt

To help you understand and recognize errors in PDDL domain files, here are examples of flawed PDDL files with explanations of the issues:

Example 1: Syntactic Error

```
(define (domain blocks)
  (:predicates (on ?x ?y)
               (ontable ?x)
               (clear ?x)
               (handempty)
               (holding ?x))

  (:action pickup
   :parameters (?x)
   :precondition (and (clear ?x) (ontable ?x) (handempty))
   :effect (and (not (ontable ?x)) (not (clear ?x)) (not (handempty)) (holding ?x))
  )
  ; Missing closing parenthesis for action definition
)
```

Error: Missing a closing parenthesis in the action definition for 'pickup'.

Example 2: Semantic Error

```
(define (domain logistics)
  (:predicates (at ?obj ?loc)
               (in ?obj ?vehicle)
               (vehicle ?vehicle)
               (location ?loc)
               (package ?obj))

  (:action load
   :parameters (?obj ?vehicle ?loc)
   :precondition (and (at ?obj ?loc) (at ?vehicle ?loc) (package ?obj) (vehicle ?vehicle))
  )
  :effect (and (not (at ?obj ?loc)) (in ?obj ?vehicle)))
  ; The precondition uses an undefined predicate (at ?vehicle ?loc)
)
```

Error: The predicate (at ?vehicle ?loc) is not defined properly and creates a semantic inconsistency.

LLM Output

Thanks for sharing these examples! I can see how these errors disrupt PDDL files: **Syntactic**

Error (Example 1): The missing closing parenthesis in the pickup action definition breaks the file structure, making it unreadable to the PDDL parser. Without proper syntax, the planning system cannot interpret actions correctly.

Semantic Error (Example 2): The predicate (at ?vehicle ?loc) is used in the load action’s precondition, but it isn’t properly defined under the predicates. This causes a logical inconsistency, as the planner cannot confirm if ?vehicle is at ?loc, leading to planning issues due to an incomplete or contradictory model.

As the LLM progresses through these phases, it receives iterative feedback on its performance, allowing it to refine its error detection and correction abilities. This iterative process is central to achieving a prompt-tuned LLM capable of handling a variety of error types across different PDDL domains. The iterative nature of prompt tuning is supported by research in [ZZH+24] and [CKD24], which emphasize the importance of feedback in improving the model’s performance over time.

5.3 Testing Dataset

Once the LLM is trained, it is tested using real-world PDDL domain models to evaluate its effectiveness in enhancing the accuracy and performance of domain models. The testing dataset comes from a comprehensive collection of PDDL instances available from GitHub-PDDL instance repository⁷. This dataset is highly relevant for testing the practical applications of LLM-enhanced domain models in real-world planning scenarios.

Relevant Features of the Testing Dataset:

Wide Variety of Problem Instances. The PDDL instances repository offers a large collection of benchmark problems from various AI planning competitions and research efforts. Domains include classical planning problems like logistics, driverlog, and rover, as well as more complex domains like numerical planning, temporal planning, and multi-agent systems. Testing on this dataset ensures that the LLM’s enhancements generalize to a broad array of planning scenarios.

Benchmark for Planners. This dataset is widely used in the AI planning community to benchmark the performance of different AI planners. By using these instances for testing, the research ensures that the LLM-based enhancements are evaluated on the same scale as traditional approaches, allowing for meaningful comparisons.

Realistic Domain Models. Unlike the flawed models used for training, the instances in this dataset represent well-constructed and realistic planning domains. The goal is to evaluate how well the LLM can refine these models to make them even more efficient or correct minor semantic issues, all while ensuring that the changes lead to improved planning outcomes.

⁷PDDL Instances GitHub repository

5 Implementation

The GitHub repository provides a structured collection of PDDL benchmark instances primarily sourced from the International Planning Competitions (IPC). Each IPC domain includes a `domain.pddl` file with a top-level `readme`, offering an informal description of the domain. Problem instances are organized in an `instances` subdirectory, named as `instance-x.pddl` (where $x \geq 1$), ensuring consistency. For some domains, specific domain descriptions are included for each instance within a `domains` subdirectory ([potassco/pddl-instances](https://github.com/potassco/pddl-instances)). The repository covers multiple IPCs with varying domains and instances:

- IPC 1998: 7 domains, 14 variants
- IPC 2000: 5 domains, 12 variants
- IPC 2002: 8 domains, 48 variants
- IPC 2004: 8 domains, 47 variants
- IPC 2006: 7 domains, 50 variants
- IPC 2008: 11 domains, 41 variants
- IPC 2011: 19 domains, 54 variants
- IPC 2014: 23 domains, 66 variants

This extensive collection serves as a valuable resource for testing and benchmarking AI planning systems, allowing for performance comparisons across different domains and problem complexities.

5.4 Domain Error Analyzing Tool

Errors—both syntactic and semantic—can significantly hinder the planner’s ability to generate valid plans and hence selecting a suitable Analyzing tool is essential. In this section, we will discuss PDDL Studio, the primary tool we have selected for Domain error analyzing task, and explain why it is vital for improving the quality of the PDDL domain models produced during the enhancement process.

5.4.1 PDDL Studio

PDDL Studio is a knowledge engineering tool designed specifically to support the creation, debugging, and refinement of PDDL models [PCBB12]. It offers a suite of features that make it invaluable for AI planners and researchers working with complex domain models. These features are particularly useful in the context of this research, where the goal is to enhance PDDL domain models using LLMs.

- Syntax Highlighting and Code Folding
- Semantic Error Detection
- Code Completion and Snippets
- Planner Integration
- Debugging Features

Feature	PDDL Studio	itSimple	PDDL-mode	MY-PDDL
syntax highlighting	yes	basic	basic	yes
semantic error detection	yes	no	no	no
code completion	yes	no	yes	yes
code snippets	no	yes	yes	yes
code folding	yes	no	yes	yes
project management	yes	no	yes	yes
planner integration	yes	yes	no	yes
interface with programming language	yes	no	no	yes
declaration menu	no	no	yes	yes
customization features	basic	no	yes	yes

Table 5.2: Comparison of Knowledge Engineering Tools and Their Features [SK20]

The Table 5.2 highlights several advanced features offered by PDDL Studio, such as syntax highlighting, semantic error detection, code completion, and planner integration, which make it an optimal choice for our implementation. Along with that PDDL Studio creates a `pddl_proj` file which acts as a centralized storage for all domain models involved in a project, consolidating both the original and refined versions. When domain models are processed—either analyzed for errors or enhanced by tools like LLMs—PDDL Studio saves these versions into the `pddl_proj` file [PCBB12].

5.4.2 Error Detection using PDDL Studio

Initiating with the installation process which is outlined in the [PCBB12]. After achieving a Prompt-Tuned Large Language Model (LLM) that has been trained to identify and resolve syntactic and semantic errors in PDDL domain files, the next crucial step is to test the PDDL domain models. To do this, we make use of PDDL Studio for detecting and analyzing PDDL errors. In this phase, we focus on testing the domain files from the aforementioned testing dataset, which consists of PDDL models from the International Planning Competitions (IPC) datasets from 1998 to 2014. Here’s a concise step-by-step outline for error detection using PDDL Studio with the testing dataset:

5 Implementation

1. **Initiate a pddl_proj File:** Begin by creating a new pddl_proj project file in PDDL Studio to serve as a centralized workspace for testing domain models.

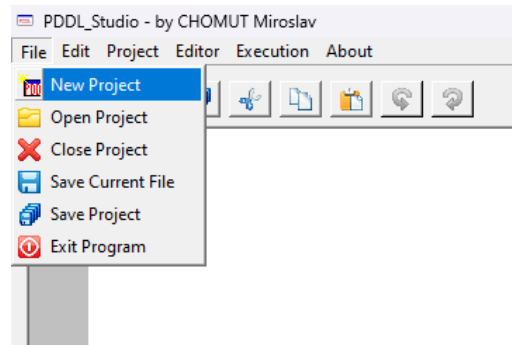


Figure 5.1: Creating new pddl_proj file

2. **Upload Testing Dataset:** Load the benchmark domain models from the testing dataset into PDDL Studio. These typically consist of various domains from IPC competitions, providing a range of syntactic and semantic error examples.

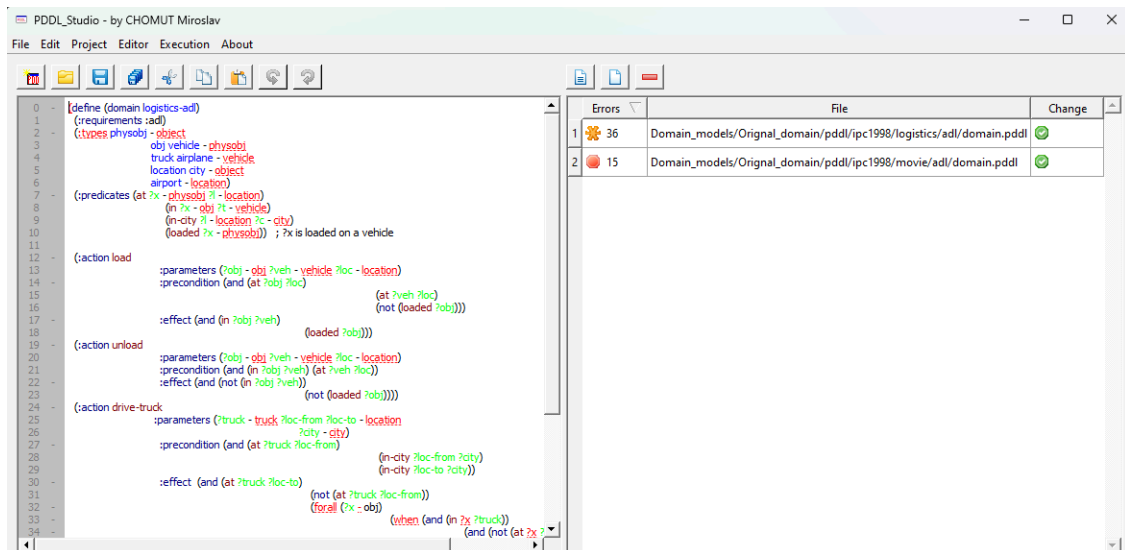


Figure 5.2: UI depicting uploaded Domain files

3. **Analyze Domain Models:** The tool will check for syntactic issues, such as missing parentheses, and semantic inconsistencies, like unsatisfied preconditions or conflicting effects.

	Chr /	Ln	Col	Origin	Error
1	1081	32	8	Effect	Conditional-effects undefined
2	1092	32	19	Parser	syntax error, unexpected T_DASH, expecting T_END or T_RBRACE or T_VAREXT
3	1109	33	10	Effect	Conditional-effects undefined
4	1123	33	24	Atomic f...	Invalid parameter
5	1158	34	23	Atomic f...	Invalid parameter
6	1185	35	12	Atomic f...	Invalid parameter
7	1251	37	25	Typed list	Typing not allowed
8	126	4	20	Typed list	Typing not allowed
9	1280	37	54	Typed list	Typing not allowed
10	1418	41	10	Effect	Conditional-effects undefined
11	1429	41	21	Parser	syntax error, unexpected T_DASH, expecting T_END or T_RBRACE or T_VAREXT
12	1448	42	12	Effect	Conditional-effects undefined
13	1462	42	26	Atomic f...	Invalid parameter
14	1499	43	25	Atomic f...	Invalid parameter
15	1528	44	14	Atomic f...	Invalid parameter

Figure 5.3: UI highlighting Error Log

4. **Review Error Report:** PDDL Studio generates a detailed error report for each analyzed file as shown in Figure 5.3, listing detected issues by type and location within the domain model.
5. **Send to Prompt-tuned LLM:** If errors are detected, the domain file is sent to the Prompt-tuned LLM as suggested in Fig .2. The LLM applies its learned corrective methods to produce an enhanced version of the domain model, free of the identified errors.

This also explains that PDDL Studio's robust PDDL parser provides real-time feedback on errors within the domain files. This feedback allows us to effectively test and validate the improvements made by the Prompt-Tuned LLM.

5.5 LLM Integration for Error Correction

After analyzing the domain models, the process flow as depicted in Figure 4.2 depends on whether errors are detected or not:

1. **If No Errors Are Detected:** In cases where PDDL Studio detects no errors (e.g., a domain model that is syntactically and semantically correct), no further intervention is required. The domain model is considered error-free, and it does not need to be processed by the Prompt-Tuned LLM.
2. **If Errors Are Detected:** As shown in the first image, errors detected in the logistics-adl and movie domain files trigger the need for further processing. When PDDL Studio flags errors, these flawed domain models are then provided as input to the Prompt-Tuned LLM for error correction.

If PDDL Studio identifies errors, the flawed domain models are sent as input to the Prompt-Tuned LLM. The LLM processes these files and generates corrected, optimized PDDL domain files. This process involves leveraging the knowledge the LLM gained during training phases mentioned in Section 4.1.2 to eliminate errors and ensure consistency.

To achieve this, we utilized two specific phases of prompts viz., Phase 3 (P3) and Phase 4 (P4) explained in Section 4.2.3. These phases are designed to assess the LLM's ability to detect errors and generate optimized, error-free PDDL domain files. Each phase builds on the previous training that the LLM underwent (Phases 1 and 2) and uses structured examples to help the model perform logical corrections

5.5.1 Testing Prompt-Tuned LLM

In phases P3 and P4, the goal is to test the IPC-benchmark domain models mentioned in Section 5.3 to evaluate the effectiveness of the trained LLM in enhancing domain model accuracy and performance. By applying the LLM to these standardized benchmarks, we assess its ability to correct or refine well-constructed, realistic models. This testing helps verify the LLM's practical impact on various complex planning scenarios and ensures its improvements generalize effectively across a diverse set of planning domains. Following are the examples of how the respective phases were implemented on Prompt-Tuned LLM.

P3 Example

LLM Prompt

Now that you have reviewed examples of flawed PDDL domain files and learned about common errors, analyze the following PDDL domain file. Identify any syntactic or semantic errors, and provide a list of detected issues with explanations.

Benchmark PDDL Domain File for Analysis: An error-prone PDDL domain file, such as the Movie domain, is presented to the LLM for analysis. This domain contains a series of predicates and actions that describe a simplified environment, with potential errors embedded in the domain model.

```
(define (domain movie-dom) (:requirements :adl :typing)
  (:types chips dip pop cheese crackers - object)
  (:predicates (movie-rewound)
               (counter-at-two-hours)
               (counter-at-zero)
               (have-chips)
               (have-dip)
               (have-pop)
               (have-cheese)
               (have-crackers))

  (:action rewind-movie
    :parameters ()
    :precondition ()
    :effect (and (movie-rewound)
                 ;; Let's assume that the movie is 2 hours long
                 (when (not (counter-at-two-hours))
                       (not (counter-at-zero))))))

  (:action reset-counter
    :parameters ()
    :precondition ()
    :effect (counter-at-zero))

  ;;; Get the food and snacks for the movie
  ....(code continues)
```

LLM Output

Here's a review of the provided movie-dom PDDL domain file, with detected syntactic and semantic errors:

Syntactic: rewind-movie action's when clause lacks proper and structure.

Semantic: reset-counter should clear (counter-at-two-hours) for consistency.

Semantic: Unused parameters in snack actions like get-chips and get-dip.

Semantic: Snack actions lack preconditions, allowing unrealistic item retrieval.

Syntactic: Extraneous quotation mark at the end disrupts file parsing.

P4 Example

LLM Prompt

Based on your understanding of PDDL domain files and the errors identified in the previous examples, generate an error-free PDDL domain file. Ensure the domain model is syntactically correct, semantically consistent, and optimized for efficient planning. Use the previous 'movie-dom' domain and ensure all predicates, actions, preconditions, and effects are logically defined.

LLM Output

Here's an optimized, error-free version of the movie-dom domain, ensuring syntactic accuracy, semantic consistency, and efficient planning:

```
(define (domain movie-dom)
  (:requirements :adl :typing)
  (:types chips dip pop cheese crackers - object)

  (:predicates
    (movie-rewound)
    (counter-at-two-hours)
    (counter-at-zero)
    (have-chips)
    (have-dip)
    (have-pop)
    (have-cheese)
    (have-crackers)
    (snack-available ?item - object))

  ;; Rewind the movie
  (:action rewind-movie
    :parameters ()
    :precondition (counter-at-two-hours)
    :effect (and
      (movie-rewound)
      (not (counter-at-two-hours))
      (counter-at-zero)))

  ;; Reset the counter to zero
  (:action reset-counter
    :parameters ()
    :precondition (not (counter-at-zero))
    :effect (and
      (counter-at-zero)
      (not (counter-at-two-hours))))

  ....(code continues)
```

5.5.2 Verification of Corrected Files

Once the Prompt-Tuned LLM produces the errorless PDDL domain models, these corrected files are re-uploaded to PDDL Studio. The tool is used again to verify whether all errors have been resolved. This ensures that the LLM-generated models are syntactically correct and semantically valid.

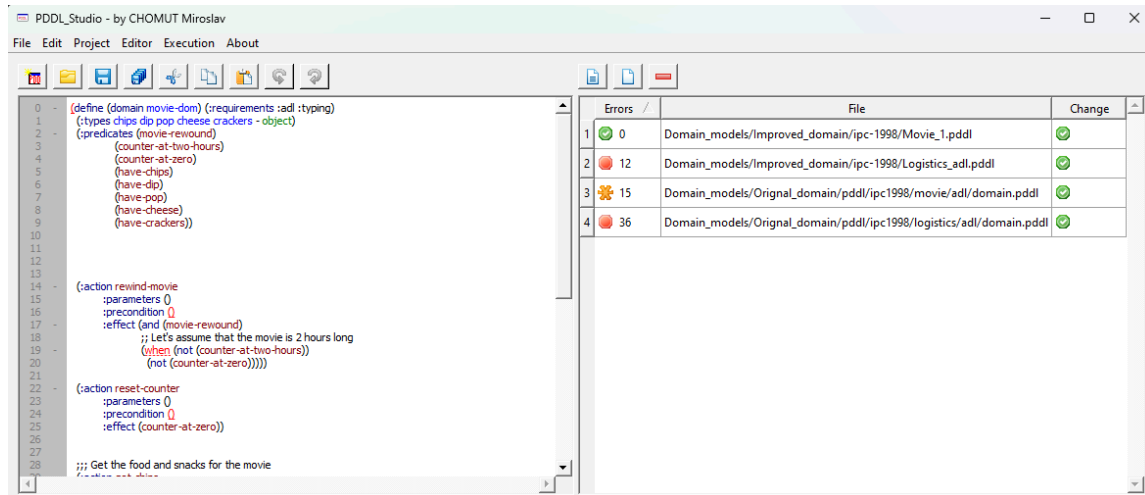


Figure 5.4: Pre- and Post-LLM Enhanced Domain files

After they are uploaded back to PDDL Studio, differentiation could be observed between the Pre- and Post-LLM enhanced PDDL Domain files as shown in Figure 5.4. The distinguishable traits observed in Figure 5.4 are:

- The movie domain file initially had 15 errors, but after the LLM’s correction, a new improved file (labeled Movie_1.pddl) is uploaded back to PDDL Studio, with 0 errors detected.
- Similarly, for the `logistics` domain, an improved domain file is generated and re-uploaded, with the error count reduced.

This iterative process ensures that errors are systematically identified and corrected until the domain files meet the required syntactic and semantic standards.

5.5.3 Output File

After the testing phase is completed, PDDL Studio generates a project file with the extension `pddl_proj`, which contains both the original error-prone domain files and the corrected versions. Each file in the `pddl_proj` file is named according to that IPC year and the domain it belongs to. For example, the tested and evaluated domain files in `ipc-1998` are in the file `ipc-1998.pddl_proj`. This file serves as a complete repository of all PDDL domain files, with clear distinctions between original and corrected versions.

5.6 AI Planner and Plan Evaluation

Once the pre-enhanced and LLM-enhanced PDDL domain files are stored in the `pddl_proj` file within PDDL Studio, the next step involves testing these models by generating plans. This process uses selected planners to create and validate action sequences, allowing for a direct comparison of plan quality across pre- and post-enhanced versions. In the following sections we will discuss about the AI planner and the metrics suitable to validate the generated plans.

5.6.1 VS-Code PDDL

The `vscode-pddl` extension plays an important role providing a comprehensive set of features to streamline PDDL modeling and automated planning within Visual Studio Code. Equipped with syntax highlighting, error detection, and snippets for key PDDL constructs [SK20], it integrates seamlessly with PDDL planners and supports both local and cloud-based configurations. These capabilities allow for faster prototyping and more efficient development of domain and problem files by eliminating some manual steps in the planning process. The application also consist of in-built Validator (VAL) which helps to evaluate the performance of plans.

The extension is accessible by searching 'PDDL' by Jan Dolejsi in VS Code's 'Extension Panel' and then installing it. For further details, refer to the [vscode-pddl GitHub page](#).

5.6.2 Temporal Fast Downward (TFD) Planner

TFD planner extends the capabilities of the original Fast Downward algorithm to handle temporal and numeric planning problems. In TFD, temporal planning is managed through a heuristic approach called the context-enhanced additive heuristic (CEA), which generalizes the original causal graph heuristic of Fast Downward [EMR12]. Additionally, TFD supports numeric features, making it versatile for domains that involve managing resources or other quantitative measures. For instance, it can handle domains like Rover, where energy levels or travel distances must be managed alongside the timing of actions. This flexibility ensures that plans generated from LLM-enhanced models are both temporally feasible and resource-efficient [HLM+19].

The TFD algorithm typically follows these high-level steps:

1. **Translation:** Convert the input PDDL task into a multi-valued planning task, which makes constraints and dependencies explicit.
2. **Heuristic Calculation:** Generate heuristics based on the causal graph of the planning task. The causal graph heuristic focuses on breaking down tasks hierarchically, helping the planner prioritize actions that are critical to achieving sub goals.
3. **Search:** Apply a forward search method using best-first search, where TFD leverages 'preferred operators' (actions that appear helpful based on the heuristic) to guide the search. TFD can manage parallel actions and adjust for action durations.
4. **Validation:** Once a solution plan is found, TFD validates temporal consistency to ensure all constraints are met.

5.6.3 Plan Evaluation process

With the assistance of VAL (plan validation system) integrated with VS-Code PDDL, various plan evaluating metrics could be computed. But in context of evaluating the performance of pre- and post-LLM enhanced domain model's generated plans, two critical metrics are introduced:

- **Makespan:** This metric represents the total time taken to complete a plan, calculated as the time from the start of the first action to the end of the last. Optimizing makespan helps ensure efficiency, especially in time-sensitive domains, by minimizing the duration of the plan [HT23].
- **Plan Cost:** Plan cost is the cumulative count of actions and resources required to achieve the plan's goal. This metric evaluates resource efficiency, with lower plan costs indicating simpler, more resource-efficient solutions that avoid unnecessary actions [RMT14].

Evaluation Process

The plan evaluation process for this implementation is structured as follows (See Appendix A.1):

1. Errored Domain Files:

- Pair each errored PDDL domain file with a corresponding problem instance file, which specifies initial conditions and goals.
- Use the TFD planner within the VSCode-PDDL extension to generate a plan based on the problem and domain files.
- Validate the generated plan using the VAL validation tool to compute metrics such as *makespan* (total time required for plan completion) and *plan cost* (total number of steps in the plan).
- Record the results of makespan and plan cost for each errored domain file.

2. Post-LLM Enhanced Domain Files:

- After LLM enhancement, pair each enhanced domain file with the same problem instance file used with the errored domain.
- Generate a plan using the TFD planner and validate it using VAL to compute the makespan and plan cost metrics.
- Store the results for each enhanced domain file for comparison.

3. Recording and Analyzing Results:

- Systematically compare the results from both the errored domain files and the post-LLM enhanced domain files.
- Analyze the makespan and plan cost to determine if LLM-enhanced domain models show improvements in terms of shorter makespan (faster plans) and lower plan cost (fewer steps).

6 Experimental Evaluation

6.1 Experimental Setup

The experimental setup of this thesis is organized into three main sections, they are as follows:

6.1.1 Technical Configuration

The technical configurations of the experimental setup are designed to optimize and assess the performance of the LLM-enhanced PDDL domain models through a series of tools for PDDL model analysis, plan generation, and validation. The setup is hosted on some hardware and software configuration which are as follows:

Hardware Configuration

- **Operating System:** Windows 11 Pro
- **Processor:** Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz
- **RAM:** 16 GB DDR5
- **GPU:** NVIDIA GeForce GTX-1050Ti

Software configuration

- **PDDL Studio:** Version 3.0, used for error detection and syntax highlighting in PDDL files.
- **VS Code:** Version 1.94.2, Visual Studio Code editor, employed for coding and integrating various tools and extensions.
- **VS Code PDDL Extension:** Version 2.22, used for running and testing PDDL plans.
- **TFD Planner:** Employed for planning and generating action sequences.
- **VAL (Validator):** used for validating plans and computing evaluation metrics like Makespan and Plan Cost.
- **LLM (GPT-4):** Direct Model Interface via ChatGPT by OpenAI (max token size = 500), used for generating and refining PDDL domain models.
- **Power BI:** 2.137.1102.0 64-bit (October 2024), Data visualization tool used for analyzing and visualizing Domain errors.

6.1.2 Scenarios

Scenario A: Analysis of Pre- and Post-LLM Error Reduction

In Scenario A, the goal is to examine the reduction of syntactic and semantic errors in the PDDL domain models after LLM enhancement. By comparing baseline models (see Section 5.3) before and after the enhancement, we aim to understand how effectively the LLM has addressed common error types that can hinder the planner’s ability to generate optimal plans. Here, we use structured Prompt phases for generating error-less PDDL Domain model and later utilizing a domain error analyzer tool as described in Section 5.5 to identify and document errors in both versions of the models. This analysis provides insights into whether LLM refinement results in a cleaner and more reliable domain model, reducing issues that could impact plan generation.

Scenario B: Measuring the Improvement in Model Quality Through Evaluation Metrics

In Scenario B, the objective is to establish a systematic approach for assessing the quality improvements introduced by the LLM-enhanced models. This scenario focuses on quantifying the effectiveness of the LLM’s corrections by analyzing changes in model quality, specifically looking at reductions in errors and enhancements in model completeness. By developing a standardized way to measure these improvements (See Section 4.3.1), we aim to create a clear picture of the LLM’s contribution to refining domain models. This structured analysis provides a consistent basis for evaluating LLM performance across various models, ensuring that the enhancement process yields measurable benefits.

Scenario C: Evaluation of Generated Plans for Effectiveness and Efficiency

In Scenario C, we evaluate the plans generated from the enhanced domain models, focusing on their effectiveness and efficiency. Here, the objective is to determine whether the LLM’s enhancements lead to more streamlined and goal-aligned plans. By analyzing the plans produced from both pre- and post-enhancement models, we aim to see if the refined models result in better task execution sequences, with an emphasis on factors like plan conciseness and goal achievement. This comparison helps identify the practical impact of LLM refinement, revealing if the improvements to the domain models translate into more optimized planning outcomes.

6.1.3 Evaluation Metrics

In this section, we outline the metrics used to assess the effectiveness of the LLM-enhanced PDDL domain models. These metrics provide both qualitative and quantitative insight into the improvements in model accuracy, planning efficiency, and overall robustness. Each metric is specifically chosen to evaluate different facets of model enhancement, from error reduction to plan optimization, ensuring a comprehensive understanding of the LLM’s impact.

Table 6.1: Evaluation Metrics for PDDL Domain Quality Assessment

Metric	Description
EPCS Score	Enhanced PDDL Correctness Score (EPCS) evaluates the improvement in syntactic and semantic accuracy of domain models after LLM enhancement see Section 4.3.1
Makespan	Represents the total time required to execute the generated plan see Section 4.3.2
Plan Cost	Indicates the overall 'cost' of the plan see Section 4.3.2

6.2 Results

6.2.1 Semantic and Syntactic Error analysis

We will discuss the three observations encountered during the evaluation of PDDL domain models with LLM assistance. Each observation represents a unique set of challenges for error detection and correction, influencing the subsequent enhancement process. The different types of scenarios illustrate varying levels of error complexity within the domain models and help outline the strengths and limitations of the LLM's capabilities in addressing these issues.

1. No Error Detection

The PDDL domain model is assessed by the error detection tool (PDDL Studio) and found to be error-free. As no syntactic or semantic errors are detected, no LLM enhancement is required, and the model is immediately ready for plan generation. This scenario serves as a baseline, demonstrating that not all domain files require refinement and helps in focusing LLM enhancement efforts on genuinely flawed models.

```

0  ; Hello World example domain
1
2  - (define (domain hello)
3
4  (:requirements :strips :typing :negative-preconditions)
5
6  (:types thing)
7
8  - (:predicates
9    (can_hear ?t - thing) ; can the `thing` hear
10   (said_hello_to ?t - thing) ; records that was said to the `thing`
11  )
12
13  ; this action greets one thing by its name
14  - (:action say-hello
15    :parameters (?t - thing)
16    :precondition (and
17      ; we only ever need to greet once
18      (not (said_hello_to ?t))
19      ; only greet someone if they are near
20      (can_hear ?t)
21    )
22    :effect (and
23      ; record that we said hello
24      (said_hello_to ?t)
25    )
26  )
27
28 )

```

Figure 6.1: Example of an errorless PDDL Domain

When provided with an error-free PDDL domain model, GPT-4 can accurately recognize the model's correctness and refrains from introducing unnecessary changes, ensuring unbiased results. This ability to distinguish between flawed and accurate models highlights GPT-4's understanding of the domain structure and allows it to apply enhancements only when errors are detected, maintaining the integrity of the original model.

LLM Output

The provided domain file for a 'Hello World' example in PDDL is error-free due to its adherence to correct syntax and semantic logic in defining the domain elements. Here's a high-level explanation of why there are no errors:

- 1. Correct Syntax:** The PDDL file follows the syntax rules of the language:
 - The domain is correctly defined within the `(define (domain hello))` structure.
 - Required PDDL features `(:strips, :typing, :negative-preconditions)` are specified under `(:requirements)`.
 - `(:types)` and `(:predicates)` are declared appropriately.
 - The `:action` block uses the correct structure for defining parameters, preconditions, and effects.
- 2. Clear Action Definition:** The action `say-hello` is defined logically:
 - **Parameters:** The action has one parameter, `?t`, typed as `thing`.
 - **Preconditions:** Preconditions ensure that the action will only execute if the specified thing can hear `((can_hear ?t))` and has not already been greeted `((not (said_hello_to ?t)))`.
 - **Effects:** The action effect `(said_hello_to ?t)` records that the greeting has occurred, without contradictions or redundant effects.
- 3. Logical Consistency:** The predicates `(can_hear ?t)` and `(said_hello_to ?t)` are simple and clear, with no logical contradictions in their relationships. The domain model logically ensures that 'things' are only greeted once, and only if they can hear, aligning with the preconditions and effects.

2. Complete Domain Model with Error Detection

In this case, the domain model is complete but contains syntactic or semantic errors identified by PDDL Studio, and hence requires further refinement. These models are then passed through the prompt-tuned LLM for error correction. Throughout our entire implementation this case particularly will be encountered and evaluated the most among all three observations.

The screenshot displays the PDDL Studio interface for a 'parking' domain. The left pane shows the PDDL code, and the right pane shows an error log with 5 errors.

```

0 (define (domain parking)
1 (requirements :strips :typing :action-costs)
2 (:types car curb)
3 (:predicates
4 (at-curb ?car - car)
5 (at-curb-num ?car - car ?curb - curb)
6 (behind-car ?car ?front-car - car)
7 (car-clear ?car - car)
8 (curb-clear ?curb - curb)
9 )
10 )
11 (:functions (total-cost) - number)
12
13 (action move-curb-to-curb
14 :parameters (?car - car ?curbsrc ?curbdest - curb)
15 :precondition (and
16 (car-clear ?car)
17 (curb-clear ?curbdest)
18 (at-curb-num ?car ?curbsrc)
19 )
20 :effect (and
21 (not (curb-clear ?curbdest))
22 (curb-clear ?curbsrc)
23 (at-curb-num ?car ?curbdest)
24 (not (at-curb-num ?car ?curbsrc))
25 )
26 (increase (total-cost) 1)
27 )
28
29 (action move-curb-to-car
30 :parameters (?car - car ?curbsrc - curb ?cardest - car)
31 :precondition (and
32 (car-clear ?car)
33 (car-clear ?cardest)
34 (at-curb-num ?car ?curbsrc)
35 (at-curb ?cardest)
36 )
37 :effect (and
38 (not (car-clear ?cardest))
39 (curb-clear ?curbsrc)
40 (behind-car ?car ?cardest)
41 (not (at-curb-num ?car ?curbsrc))
42 (not (at-curb ?car))
43 )
44 (increase (total-cost) 1)
45 )

```

Errors /	File
1	Domain_models/Improved_domain/ipc-2011/parking.pddl
2	Domain_models/Original_domain/pddl-instances/ipc-2011/domains/parking-sequential-multi-core/domain.pddl

Figure 6.2: Example of Parking PDDL Domain

The screenshot displays the PDDL Studio interface for a 'pegsolitaire' domain. The left pane shows the PDDL code, and the right pane shows an error log with 6 errors.

```

0 (define (domain pegsolitaire-sequential)
1 (requirements :typing :action-costs)
2 (:types location - object)
3 (:predicates
4 (IN-LINE ?x ?y ?z - location)
5 (occupied ?l - location)
6 (free ?l - location)
7 (move-ended)
8 (last-visited ?l - location)
9 )
10 (:functions (total-cost) - number)
11
12 (action jump-new-move
13 :parameters (?from - location ?over - location ?to - location)
14 :precondition (and
15 (move-ended)
16 (IN-LINE ?from ?over ?to)
17 (occupied ?from)
18 (occupied ?over)
19 (free ?to)
20 )
21 :effect (and
22 (not (move-ended))
23 (not (occupied ?from))
24 (not (occupied ?over))
25 (occupied ?to)
26 (not (free ?to))
27 (free ?from)
28 (free ?over)
29 (last-visited ?to)

```

Errors /	File
1	Domain_models/Improved_domain/ipc-2011/pegsolitaire.pddl
2	Domain_models/Original_domain/pddl-instances/ipc-2011/domains/peg-solitaire-sequential-multi-core/

Figure 6.3: Example of pegsolitaire PDDL Domain

In Figure 6.2 and Figure 6.3 we can observe that the number of errors are reduced successfully Post-LLM Enhancement of 'Parking' domain. This scenario is central to evaluating the LLM's capability to reduce error counts and improve model quality, particularly regarding syntactic and certain semantic issues.

3. Incomplete Domain Models

This challenging scenario involves domain models that are both incomplete and contain errors, such as missing actions or predicates necessary for defining all problem requirements. Due to the incomplete nature of the model, even after LLM enhancement, error reduction remains limited or entirely ineffective. The persistent issues suggest that while the LLM can rectify certain syntactic and semantic errors, it struggles with more complex structural deficiencies that require deeper domain knowledge or additional contextual input.

	Chr	Ln	Col	Origin /	Error
1	516	15	1	Parser	syntax error, unexpected T_COLONTEXT
2	1127	27	1	Parser	syntax error, unexpected T_COLONTEXT
3	1126	27	0	Domain	Multiple incomplete parts
4	1353	33	1	Parser	syntax error, unexpected T_COLONTEXT
5	1352	33	0	Domain	Multiple incomplete parts
6	1531	39	1	Parser	syntax error, unexpected T_COLONTEXT
7	1530	39	0	Domain	Multiple incomplete parts
8	2131	49	1	Parser	syntax error, unexpected T_COLONTEXT
9	2130	49	0	Domain	Multiple incomplete parts
10	2900	60	1	Parser	syntax error, unexpected T_COLONTEXT
11	2899	60	0	Domain	Multiple incomplete parts

Figure 6.4: Error log for an Incomplete Domain Model

The screenshot displays a domain model editor with a code editor on the left and an error log on the right. The code editor shows several actions defined in a domain model, including 'choose', 'initialize', 'associate', and 'associate-with-catalyze'. The error log on the right shows two errors, both with a count of 11. The first error is a syntax error in the file 'Domain_models/Improved_domain/ipc-2006/Pathways.pddl', and the second error is a domain error in the file 'Domain_models/Original_domain/pddl-instances/ipc-2006/...'. Both errors are marked with a green checkmark, indicating they have been resolved or are not critical.

Figure 6.5: Example of 'Pathways' Domain Model

In the Figure 6.4, error logs indicate multiple instances of 'syntax error, unexpected T_COLONTEXT' and 'multiple incomplete parts', which highlight the absence of essential components required for a valid domain structure.

The Figure 6.5 shows the highlighted areas in red, visually marking the problematic sections within the PDDL domain file, where incomplete specifications, such as missing conditions and effects in durative actions, render the model non-functional. We can also see there are no reduction in errors even post-LLM enhancement of the incomplete domain model.

The persistent 'multiple incomplete parts' issue exemplifies how certain domain models require comprehensive human intervention for structural completion before they can undergo effective error correction and validation by an automated system. This underscores the limitations of LLMs in handling deeply flawed or incomplete models and emphasizes the need for complete domain specifications for meaningful enhancements.

Based on the three primary observations, following are the data tables which highlights those observations and depicts various error reductions and trends in ipc-baseline PDDL Domain models:

Table 6.2: Semantic and Syntactic Errors Pre and Post LLM-Enhancement

PDDL Domain Files		Pre LLM-Enhancement		Post LLM-Enhancement	
		Semantic	Syntactic	Semantic	Syntactic
ipc-1998	Logistics	14	22	10	2
	Assembly	58	18	40	10
	Movie	8	7	0	0
ipc-2000	Elevator	83	22	52	0
	Schedule	80	17	10	5
	Freecell	19	11	8	1

The Table 6.2 represents the Pre and Post LLM-Enhancement error analysis for PDDL domain files from two IPC datasets: ipc-1998 and ipc-2000. The results show a reduction in both semantic and syntactic errors after the LLM enhancement. For each domain, the semantic errors decrease, and syntactic errors are almost entirely eliminated in most cases.

Table 6.3: Semantic and Syntactic Errors Pre and Post LLM-Enhancement (ipc-2002)

ipc-2002	Pre LLM-Enhancement		Post LLM-Enhancement	
	Semantic	Syntactic	Semantic	Syntactic
Zenotravel_strips	12	1	7	0
Zenotravel_time-auto (incomplete domain)	7	5	7	5
Depot_time-auto (incomplete domain)	6	6	6	6
Driverlog_strips	8	4	0	0
Driverlog_time-hand-code (incomplete domain)	7	6	7	6
Rovers_strips	22	8	12	2
Rovers_time-auto (incomplete domain)	10	11	10	9
Satellite_strips	4	1	0	0
Satellite_time-automatic (incomplete domain)	5	6	5	6

The Table 6.3 highlights the Pre and Post LLM-Enhancement error analysis for PDDL domain files from the ipc-2002 dataset, including both complete and incomplete domains. The results show that syntactic errors are largely reduced or eliminated across most domains. However, semantic errors are not removed as efficiently. For instance, in domains such as *Rovers_strips* and *Zenotravel_strips*, while syntactic errors were completely removed, a noticeable number of semantic errors remain. Furthermore, for incomplete domains (denoted in the table with 'incomplete domain' like *Zenotravel_time-auto* and *Depot_time-auto*), the LLM was unable to improve these files.

Table 6.4: Semantic and Syntactic Errors Pre and Post LLM-Enhancement (ipc-2004)

ipc-2004	Pre LLM-Enhancement		Post LLM-Enhancement	
	Semantic	Syntactic	Semantic	Syntactic
Settlers_strips	103	161	90	120
Pipesworld_strips	7	6	4	0
Pipesworld_temp_strips (incomplete domain)	5	8	5	7
Promela	23	6	13	0
PSR_adl	20	26	12	8
PSR (incomplete domain)	15	7	15	5
UMTS_strips	9	10	5	4
UMTS_temp_strips (incomplete domain)	10	9	10	8
Airport_adl (incomplete domain)	56	84	56	70
Airport_temp_adl	12	31	6	0
Satellite_time_strips (incomplete domain)	8	4	8	3

The data in the Table 6.4 indicates that the LLM enhancement has effectively reduced both semantic and syntactic errors across multiple PDDL domain files. For instance, in the *Settlers_strips* domain, semantic errors dropped from 103 to 90, and syntactic errors from 161 to 120, reflecting the LLM's positive impact on improving domain file accuracy and consistency.

Table 6.5: Semantic and Syntactic Errors Pre and Post LLM-Enhancement (ipc-2006)

ipc-2006	Pre LLM-Enhancement		Post LLM-Enhancement	
	Semantic	Syntactic	Semantic	Syntactic
Pathways_adl	5	8	3	2
Pathways (incomplete domain)	5	6	5	6
Pipesworld_metric-time	7	6	2	3
Pipesworld_preferences (incomplete domain)	7	6	7	6
Rovers_metric-simple	3	1	0	0
Rovers_metric-time	10	11	10	10
Storage_pref-complex (incomplete domain)	5	5	5	5
Storage_propositional	0	0	0	0
tpp-metric	12	21	10	12
tpp_preferences-complex (incomplete domain)	7	6	7	5
Openstacks_metric-time (incomplete domain)	4	5	4	5
Trucks_time (incomplete domain)	5	6	5	4

Table 6.6: Semantic and Syntactic Errors Pre and Post LLM-Enhancement (ipc-2008)

ipc-2008	Pre LLM-Enhancement		Post LLM-Enhancement	
	Semantic	Syntactic	Semantic	Syntactic
Crewplan_numeric-fluents	27	58	19	10
Elevator_numeric-fluents	5	14	3	8
Elevator_optimal-strips (incomplete domain)	4	9	4	9
Model-train	11	10	8	3
Openstacks_numeric-fluents	9	9	5	3
Openstacks_num-fluents-adl (incomplete domain)	3	4	3	4
peg-solitaire	2	0	0	0
scanalyzer-3d-strips	6	7	3	2
Sokoban_strips	5	11	4	5
Transport	8	4	4	2
Woodworking	14	18	9	8

Table 6.7: Semantic and Syntactic Errors Pre and Post LLM-Enhancement (ipc-2011)

ipc-2011	Pre LLM-Enhancement		Post LLM-Enhancement	
	Semantic	Syntactic	Semantic	Syntactic
Barman-multi-core	13	17	9	12
Crewplan (incomplete domain)	14	15	14	10
Elevator-multi-core	4	9	0	3
Elevator-satisficing (incomplete domain)	6	7	6	7
Floortile-multi-core	10	8	7	4
Floortile-satisficing (incomplete domain)	7	6	7	6
match-cellar (incomplete domain)	1	2	1	2
no-mystery-optimal	4	4	0	2
Parking-optimal	5	7	3	3
Parking-satisficing (incomplete domain)	4	3	4	3
peg-solitaire	2	0	1	1
machine-shop (incomplete domain)	9	9	10	7
Transport-optimal	6	2	2	2
Woodworking-multi-core	15	19	12	11
Turn-and-open (incomplete domain)	4	5	4	5

Table 6.8: Semantic and Syntactic Errors Pre and Post LLM-Enhancement (ipc-2014)

ipc-2014	Pre LLM-Enhancement		Post LLM-Enhancement	
	Semantic	Syntactic	Semantic	Syntactic
City-car	11	12	8	7
Driver-log (incomplete domain)	5	6	5	6
Floor-Tile	3	13	2	9
Hiking	3	0	0	0
Maintenance	3	0	0	0
Map-analyzer (incomplete domain)	6	5	6	5
Match-cellar (incomplete domain)	1	2	1	2
Tetris	10	12	7	10
Transport	5	5	3	4
Road-traffic-accident-mgmt (incomplete domain)	11	12	11	12
Parking	7	6	5	2

The results from ipc-2006, ipc-2008, ipc-2011 and ipc-2014 demonstrate a trend where the LLM enhancement is effective at reducing syntactic errors, but the reduction in semantic errors is less pronounced across several domains.

6.2.2 LLM Evaluation using EPCS

To assess the overall effectiveness of the LLM enhancements, a custom metric called the EPCS has been developed as mentioned in Section 4.3.1. This metric quantifies the level of error reduction in the enhanced models, considering both syntactic and semantic errors before and after LLM correction. The EPCS score offers a clear numerical indicator of the success of the LLM-enhanced domain models.

Table 6.9: EPCS Metrics for PDDL Domain Files

PDDL Domain File	EPCS Metric
LOGISTICS	66.67
ASSEMBLY	34.21
MOVIE	100.00
ELEVATOR	50.47
SCHEDULE	84.53
FREECCELL	70.00
ZENOTRAVEL	46.15
DRIVERLOG	100.00
ROVERS	53.33
SATELLITE	100.00
SETTLERS	20.45
PIPESWORLD	69.23
PROMELA	56.67
PSR	56.52
UMTS	55.00
AIRPORT	86.00
PATHWAYS	64.30
ROVERS (2006)	100.00
STORAGE	100.00
TPP	33.33
CREWPLAN	65.88
MODEL-TRAIN	52.40
OPENSTACKS	55.55
PEG-SOLITAIRE	100.00
SCANALYZER-3D	61.53
SOKOBAN	55.00
TRANSPORT	52.63
WOODWORKING	43.33
BARMAN	30.00
ELEVATOR (2011)	76.92
FLOOR-TILE	38.89
NO-MYSTERY	75.00
PARKING	53.84

Continued on next page

PDDL Domain File	EPCS Metric
MACHINE-SHOP	10.53
PEG-SOLITAIRE (2011)	75
CITY-CAR	34.78
FLOOR-TILE (2014)	31.25
HIKING	100.78
TETRIS	22.72
TRANSPORT (2014)	30.00

The Table 6.9 of EPCS scores presents a broad range of results across various domain files, providing valuable insights into the effectiveness of LLM-based enhancement for improving PDDL domain models. A few trends and observations emerge from the data:

- **High EPCS Scores (Near or Equal to 100):** Several domain models, such as *Movie*, *Driverlog*, *Rovers (2006)*, *Satellite*, and *PEG-Solitaire*, achieve perfect scores of 100, indicating that the LLM has successfully removed all or most errors, bringing these domains to an error-free or near-optimal state. This suggests that for certain domains, the LLM is highly effective in addressing both syntactic and semantic errors.
- **Moderate Scores (50-85 Range):** A significant number of domains fall within this range, such as *Logistics*, *Elevator*, *Pipesworld*, and *Crewplan*. These scores indicate that while the LLM has made improvements, some residual issues or errors remain. This suggests that for certain domains, the LLM may not completely eliminate all errors, but it still improves the overall correctness and quality of the domain models.
- **Low EPCS Scores (Below 50):** Domains such as *Assembly*, *Settlers*, and *Barman* show relatively low scores, indicating that the LLM was less effective in correcting errors. These lower scores could point to more complex domain-specific challenges that the LLM might not fully understand or handle effectively. It may also indicate that certain types of errors are more resistant to correction.
- **Incomplete Domain Models:** It's important to note that certain domain models, particularly incomplete ones, may be more challenging for the LLM to enhance. For example, *Machine-Shop* has an extremely low score of 10.53, highlighting that despite the LLM's efforts, significant errors remain unresolved. Also, in all cases where incomplete domains are observed, the EPCS score is '0' and hence those are not mentioned in the table.

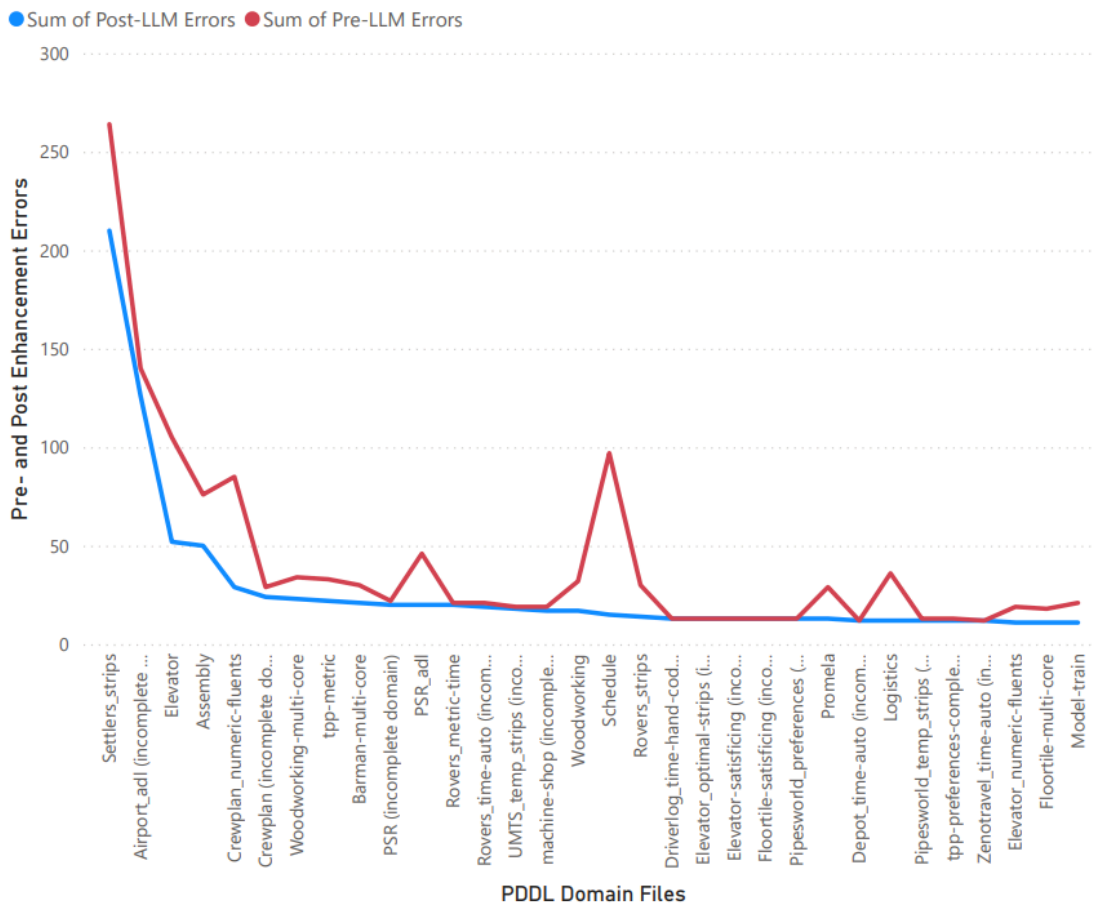


Figure 6.6: Pre- and Post-LLM Enhanced Error trend

From Figure 6.6, it's clear that in most domain files, the LLM-enhanced models demonstrate a reduction in errors compared to their initial states, indicated by the blue line generally staying below the red line. The effectiveness of the LLM is evident in the significant drop in errors for several domains, reflecting the EPCS score, which measures the percentage improvement after enhancement. Domains with a large gap between pre- and post-LLM errors suggest a higher EPCS score, indicating substantial error correction, while domains with minimal differences or persistent errors indicate areas where LLM enhancement was less effective.

6.2.3 Validation of Plan

After Plan evaluation this scenario could be differentiated into two more categories:

1. Plan Metric improvements

In Figure 6.2 we can observe that the number of errors are reduced successfully Post-LLM Enhancement of Parking domain and hence it undergoes AI plan generation and evaluation.

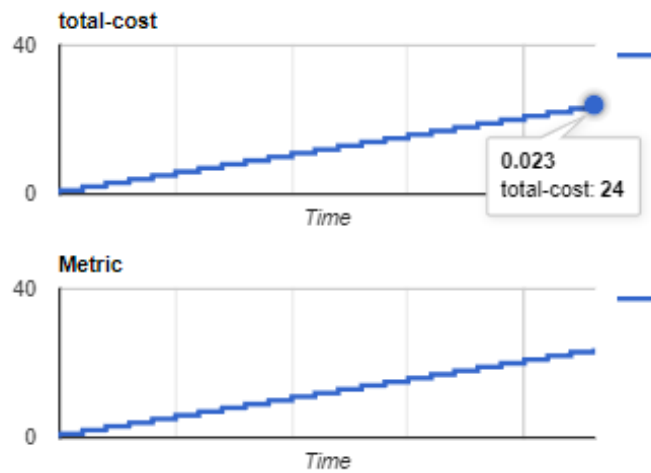


Figure 6.7: Parking domain Pre-LLM Enhancement metric visualization

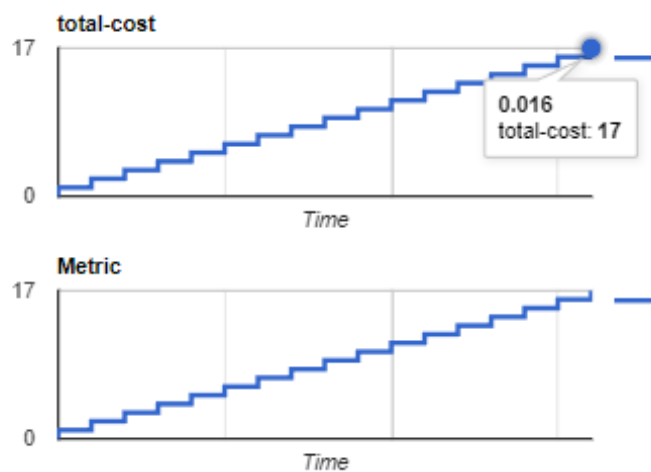


Figure 6.8: Parking domain Post-LLM Enhancement metric visualization

We can observe in Figure 6.7 and Figure 6.8 the cost for the Pre-LLM Enhanced plan is '24' while the cost for Post-LLM Enhanced plan is '17' which is a significant improvement. Along with that, Makespan has been reduced from 0.023 to 0.016, which is not huge difference. But even smaller improvements are critical as discussed in Section 4.3.2.

2. No Plan Metric improvements

If we consider pegsolitaire domain as seen in Figure 6.3, we encounter another case where no improvements are observed in plan evaluation metrics even after there have been significant reduction.

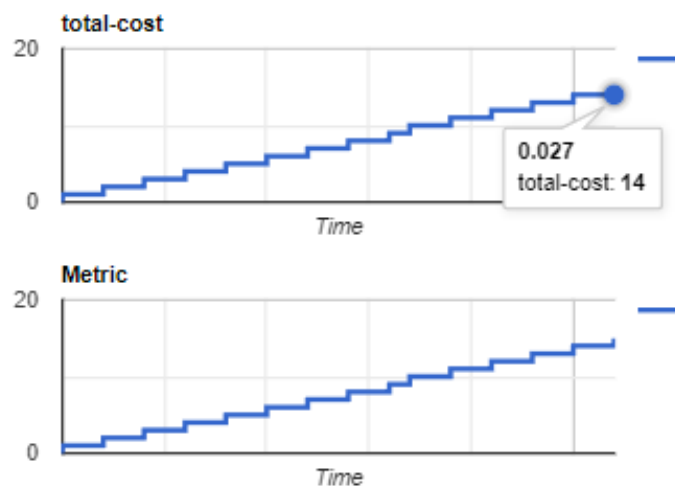


Figure 6.9: pegsolitaire domain Pre-LLM Enhancement metric visualization

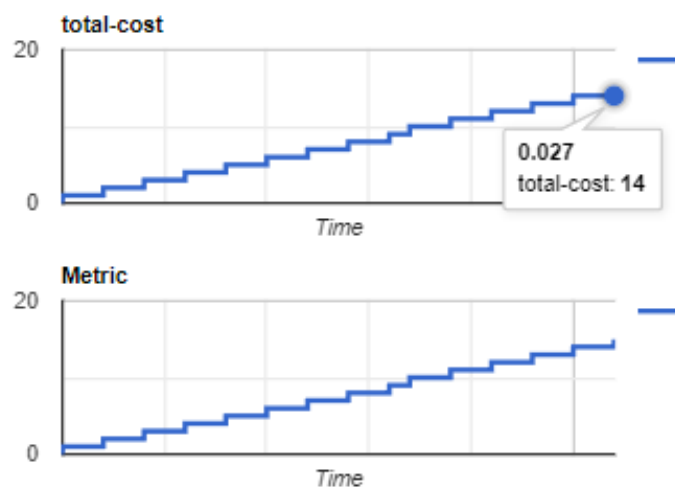


Figure 6.10: pegsolitaire domain Post-LLM Enhancement metric visualization

In Figure 6.9 and Figure 6.10 we can see no improvements in Makespan as well as Plan cost has been observed, unlike in Parking domain example where visible enhancements are observed.

Table 6.10: Evaluation of the generated plan Pre- and Post-LLM Enhancement

PDDL Domain Files	Errored PDDL Domain		LLM Enhanced PDDL Domain	
	Makespan	Plan Cost	Makespan	Plan Cost
Depot	59.00278	13	57.74278	11
Driverlog	302.07	8	300.63	7
PSR	0.005	6	0.005	6
Pipesworld	-	6	12.05	6
Rover	311.6	9	309.43	7
Storage	-	3	0.002	3
Crewplan	2836.109	12	2440.01	10
Elevators	160.18	18	160.18	18
Model-train	-	-	463.2	33
Wood-working	-	-	90.021	3
Settlers	0.001	2	0.001	2
Parking	0.023	17	0.016	17
Peg-solitaire	0.027	15	0.026	14

The Table 6.10 highlights the validated results and depicts aforementioned two cases for some of the benchmark PDDL Domain models. The results presented in the table Table 6.10 a clear improvement in both makespan and plan cost for most of the domain files after LLM enhancement of the PDDL domain models. For example, in the Depot domain, the makespan decreased from 59.00278 to 57.74278, and the plan cost from 13 to 11, showing how the LLM improvement resulted in a more efficient plan. Similar improvements are observed in the Driverlog and Rover domains, where makespan and plan cost values are reduced. This indicates that the LLM-enhanced domain files result in more optimized plans, possibly by removing syntactic and semantic errors that were causing inefficiencies in the planning process.

Some domain files like Pipesworld, Storage, and Model-train did not generate plans in the errored domain files (indicated by '-'), but were able to generate plans after the LLM enhancement, with reasonable makespan and plan cost values. This demonstrates that the LLM successfully addressed the critical errors that previously prevented plan generation, allowing the planners to produce valid solutions. For example, in the Model-train domain, the LLM enhancement resulted in a makespan of 463.2, a significant improvement given that no plan could be generated before.

However, not all domains exhibit the same level of improvement. For instance, in the Elevators domain, the makespan and plan cost remained the same between the errored and enhanced domain files, suggesting that the LLM's ability to improve these specific domain files may have been limited. This may point to certain types of errors that the LLM struggled to address fully, or perhaps the domain itself inherently presents complexities that are harder to optimize. Nonetheless, the overall trend shows a positive impact from LLM enhancements, particularly for domains with severe initial errors.

7 Conclusion

This research's primary outcome is the investigation and partial improvement of domain models for AI planning utilizing LLMs. Utilizing GPT-4 in a prompt-tuned configuration, we sought to automate the identification and rectification of syntactic and semantic faults in PDDL domain files, a prevalent impediment in AI planning. While the LLM had a significant ability to identify and rectify syntactic problems, its efficacy in addressing semantic inconsistencies was inconsistent. The EPCS metric presented in this thesis enables the quantification of enhancements in domain model accuracy, providing a valuable metric for assessing LLM-augmented domain models. This result, however encouraging, illustrates the capabilities and constraints of contemporary LLMs in improving domain models inside intricate settings.

The results indicate that although LLMs, notably GPT-4, can effectively rectify syntactic errors, they have difficulties with complex semantic problems, particularly in incomplete domain files. The results clearly indicated that certain domains exhibited substantial enhancements in EPCS scores following LLM augmentation, whereas others displayed negligible or no improvement in aspects such as 'multiple incomplete parts' and undefined parameters, which necessitate advanced logical reasoning and contextual comprehension. Nonetheless, our findings signify progress in automated PDDL domain augmentation and suggest that LLMs can function as beneficial instruments in the domain engineering process when integrated with conventional error-checking systems such as PDDL Studio.

We have successfully demonstrated that LLMs may be effectively integrated into the PDDL domain improvement pipeline for automated error detection and repair procedures. The degree of success is constrained by the intricacy of certain mistake kinds. The study emphasizes the existing constraints of LLMs in resolving profound logical discrepancies, necessitating enhanced domain-specific knowledge and reasoning capabilities. This discovery is essential since it delineates the limits of existing LLM capabilities and identifies areas requiring human or more sophisticated AI involvement.

Prospectively, several pathways for more research exist. A potential avenue is to investigate hybrid systems that integrate LLMs with symbolic AI or constraint-solving methodologies, which may enhance the management of intricate semantic mistakes. Additional refinement of the LLM using domain-specific training data may enhance its comprehension of planning-specific elements and logical linkages inside domain models. Furthermore, more sophisticated assessment metrics might be established to more effectively capture the subtleties of domain file enhancements, transcending basic syntactic and semantic mistake correction to include metrics that evaluate planning efficiency and flexibility.

This thesis has advanced the automation of PDDL domain augmentation through the use of LLMs; nonetheless, it is evident that LLMs alone may not provide the ultimate answer. They can serve as a supplementary instrument that operates in conjunction with human specialists or more advanced AI systems. Further exploration of the integration of LLMs with AI planning tools holds the potential to diminish the human labor required for domain model construction and validation, hence enhancing the efficiency and scalability of AI planning systems.

Bibliography

- [AHK+98] C. Aeronautiques, A. Howe, C. Knoblock, I. D. McDermott, A. Ram, M. Veloso, D. Weld, D. W. Sri, A. Barrett, D. Christianson, et al. “Pddl the planning domain definition language”. In: *Technical Report, Tech. Rep.* (1998) (cit. on p. 15).
- [BBF+14] O. Bojar, C. Buck, C. Federmann, B. Haddow, P. Koehn, J. Leveling, C. Monz, P. Pecina, M. Post, H. Saint-Amand, et al. “Findings of the 2014 workshop on statistical machine translation”. In: *Proceedings of the ninth workshop on statistical machine translation*. 2014, pp. 12–58 (cit. on p. 36).
- [BSA23] A. Berti, D. Schuster, W. M. van der Aalst. “Abstractions, scenarios, and prompt definitions for process mining with LLMS: a case study”. In: *International Conference on Business Process Management*. Springer. 2023, pp. 427–439 (cit. on p. 28).
- [CKD24] W. Chen, S. Koenig, B. Dilkina. “RePrompt: Planning by Automatic Prompt Engineering for Large Language Models Agents”. In: *arXiv preprint arXiv:2406.11132* (2024) (cit. on pp. 35, 43).
- [EBSD21] S. Esmacili, B. Brubach, A. Srinivasan, J. Dickerson. “Fair clustering under a bounded cost”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 14345–14357 (cit. on p. 36).
- [EDS23] T. Eckstein, M. De Bortoli, G. Steinbauer-Wagner. “Fixing PDDL Domains Using Delta Debugging”. In: *34th International Workshop on Principle of Diagnosis*. 2023 (cit. on p. 30).
- [EMR12] P. Eyerich, R. Mattmüller, G. Röger. “Using the context-enhanced additive heuristic for temporal and numeric planning”. In: *Towards Service Robots for Everyday Environments: Recent Advances in Designing Service Robots for Complex Tasks in Everyday Environments*. Springer, 2012, pp. 49–64 (cit. on p. 52).
- [Geo23] I. Georgievski. “Software Development Life Cycle for Engineering AI Planning Systems.” In: *ICSOFT*. 2023, pp. 751–760 (cit. on pp. 15, 27).
- [Geo25] I. Georgievski. “Engineering AI Planning Systems”. Habilitation Thesis. 2025 (cit. on p. 15).
- [GKS24] E. Gestrin, M. Kuhlmann, J. Seipp. “NL2Plan: Robust LLM-Driven Planning from Minimal Text Descriptions”. In: *arXiv preprint arXiv:2405.04215* (2024) (cit. on pp. 33, 35).
- [GNT04] M. Ghallab, D. Nau, P. Traverso. *Automated Planning: theory and practice*. Elsevier, 2004 (cit. on p. 21).
- [GVSK23] L. Guan, K. Valmeekam, S. Sreedharan, S. Kambhampati. “Leveraging pre-trained large language models to construct and utilize world models for model-based task planning”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 79081–79094 (cit. on pp. 16, 28, 30).

Bibliography

- [HLM+19] P. Haslum, N. Lipovetzky, D. Magazzeni, C. Muise, R. Brachman, F. Rossi, P. Stone. *An introduction to the planning domain definition language*. Vol. 13. Springer, 2019 (cit. on pp. 21, 36, 52).
- [HT23] V. N. Hartmann, M. Toussaint. “Towards computing low-makespan solutions for multi-arm multi-task planning problems”. In: *arXiv preprint arXiv:2305.17527* (2023) (cit. on pp. 37, 53).
- [KSV+13] M. R. Khouadjia, M. Schoenauer, V. Vidal, J. Dréo, P. Savéant. “Pareto-Based Multiobjective AI Planning”. In: *IJCAI 2013*. Ed. by F. Rossi. Proceedings of the 23rd International Joint Conference on Artificial Intelligence. IJCAI/AAAI. Beijing, China: IJCAI/AAAI, Aug. 2013, pp. 2321–2327. URL: <https://inria.hal.science/hal-00835003> (cit. on pp. 37, 38).
- [KVG+24] S. Kambhampati, K. Valmeekam, L. Guan, K. Stechly, M. Verma, S. Bhambri, L. Saldyt, A. Murthy. “LLMs Can’t Plan, But Can Help Planning in LLM-Modulo Frameworks”. In: *arXiv preprint arXiv:2402.01817* (2024) (cit. on pp. 17, 30).
- [LJZ+23] B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, P. Stone. “Llm+ p: Empowering large language models with optimal planning proficiency”. In: *arXiv preprint arXiv:2304.11477* (2023) (cit. on p. 16).
- [LLLJ23] J. Li, G. Li, Y. Li, Z. Jin. “Structured chain-of-thought prompting for code generation”. In: *ACM Transactions on Software Engineering and Methodology* (2023) (cit. on p. 32).
- [McC03] T. L. McCluskey. “PDDL: A Language with a Purpose?” In: (2003) (cit. on p. 15).
- [MSF+23] L.-P. Meyer, C. Stadler, J. Frey, N. Radtke, K. Junghanns, R. Meissner, G. Dziwis, K. Bulert, M. Martin. “Llm-assisted knowledge graph engineering: Experiments with chatgpt”. In: *Working conference on Artificial Intelligence Development for a Resilient and Sustainable Tomorrow*. Springer Fachmedien Wiesbaden Wiesbaden. 2023, pp. 103–115 (cit. on p. 28).
- [MVV16] T. L. McCluskey, T. Vaquero, M. Vallati. “Issues in planning domain model engineering”. In: (2016) (cit. on p. 27).
- [MVV17] T. L. McCluskey, T. S. Vaquero, M. Vallati. “Engineering knowledge for automated planning: Towards a notion of quality”. In: *Proceedings of the 9th Knowledge Capture Conference*. 2017, pp. 1–8 (cit. on p. 17).
- [OSK+24] J. Oswald, K. Srinivas, H. Kokel, J. Lee, M. Katz, S. Sohrabi. “Large Language Models as Planning Domain Generators”. In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 34. 2024, pp. 423–431 (cit. on p. 30).
- [PCBB12] T. Plch, M. Chomut, C. Brom, R. Barták. “Inspect, edit and debug PDDL documents: Simply and efficiently with PDDL studio”. In: *System Demonstrations and Exhibits at ICAPS* (2012), pp. 15–18 (cit. on pp. 24, 44, 45).
- [PMR+24] V. Pallagani, B. C. Muppasani, K. Roy, F. Fabiano, A. Loreggia, K. Murugesan, B. Srivastava, F. Rossi, L. Horesh, A. Sheth. “On the prospects of incorporating large language models (llms) in automated planning and scheduling (aps)”. In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 34. 2024, pp. 432–444 (cit. on pp. 16, 28).
- [REM08] G. Röger, P. Eyerich, R. Mattmüller. “TFD: A numeric temporal extension to Fast Downward”. In: *International Planning Competition (IPC), Sydney, Australia* (2008), p. 114 (cit. on p. 19).

- [RMT14] N. Robinson, S. McIlraith, D. Toman. “Cost-based query optimization via AI planning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 28. 1. 2014 (cit. on pp. 37, 53).
- [SB23] K. Sleath, P. Bercher. “Detecting AI Planning Modelling Mistakes–Potential Errors and Benchmark Domains”. In: *Pacific Rim International Conference on Artificial Intelligence*. Springer. 2023, pp. 448–454 (cit. on pp. 16, 23).
- [SDS+24] T. Silver, S. Dan, K. Srinivas, J. B. Tenenbaum, L. Kaelbling, M. Katz. “Generalized planning in pddl domains with pretrained large language models”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 18. 2024, pp. 20256–20264 (cit. on pp. 16, 17, 29).
- [SHS+22] T. Silver, V. Hariprasad, R. S. Shuttleworth, N. Kumar, T. Lozano-Pérez, L. P. Kaelbling. “PDDL planning with pretrained large language models”. In: *NeurIPS 2022 foundation models for decision making workshop*. 2022 (cit. on pp. 19, 21, 25).
- [SJCG24] P. Smirnov, F. Joubin, A. Ceravola, M. Gienger. “Generating consistent PDDL domains with Large Language Models”. In: *arXiv preprint arXiv:2404.07751* (2024) (cit. on p. 29).
- [SK20] V. Strobel, A. Kirsch. “MyPDDL: Tools for efficiently creating PDDL domains and problems”. In: *Knowledge Engineering Tools and Techniques for AI Planning*. Springer, 2020, pp. 67–90 (cit. on pp. 45, 52).
- [VCMH21] M. Vallati, L. Chrupa, T. L. McCluskey, F. Hutter. “On the importance of domain model configuration for automated planning engines”. In: *Journal of Automated Reasoning* 65.6 (2021), pp. 727–773 (cit. on pp. 16, 17).
- [VHCM15] M. Vallati, F. Hutter, L. Chrupa, T. L. McCluskey. “On the effective configuration of planning domain models”. In: *International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI press. 2015 (cit. on pp. 21, 29).
- [VSM+23] K. Valmееkam, S. Sreedharan, M. Marquez, A. Olmo, S. Kambhampati. “On the planning abilities of large language models (a critical investigation with a proposed benchmark)”. In: *arXiv preprint arXiv:2302.06706* (2023) (cit. on p. 29).
- [WWS+22] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al. “Chain-of-thought prompting elicits reasoning in large language models”. In: *Advances in neural information processing systems* 35 (2022), pp. 24824–24837 (cit. on p. 25).
- [XYJ+23] D. Xie, B. Yoo, N. Jiang, M. Kim, L. Tan, X. Zhang, J. S. Lee. “Impact of large language models on generating software specifications”. In: *arXiv preprint arXiv:2306.03324* (2023) (cit. on pp. 15, 27).
- [ZZH+24] T. Zhang, L. Zhang, Z. Hou, Z. Wang, Y. Gu, P. Clark, C. Callison-Burch, N. Tandon. “Proc2pddl: Open-domain planning representations from texts”. In: *arXiv preprint arXiv:2403.00092* (2024) (cit. on p. 43).
- [ZZL+23] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, et al. “A survey of large language models”. In: *arXiv preprint arXiv:2303.18223* (2023) (cit. on p. 25).

All links were last followed on November 04, 2024.

A Appendix

A.1 Plan Evaluation Process

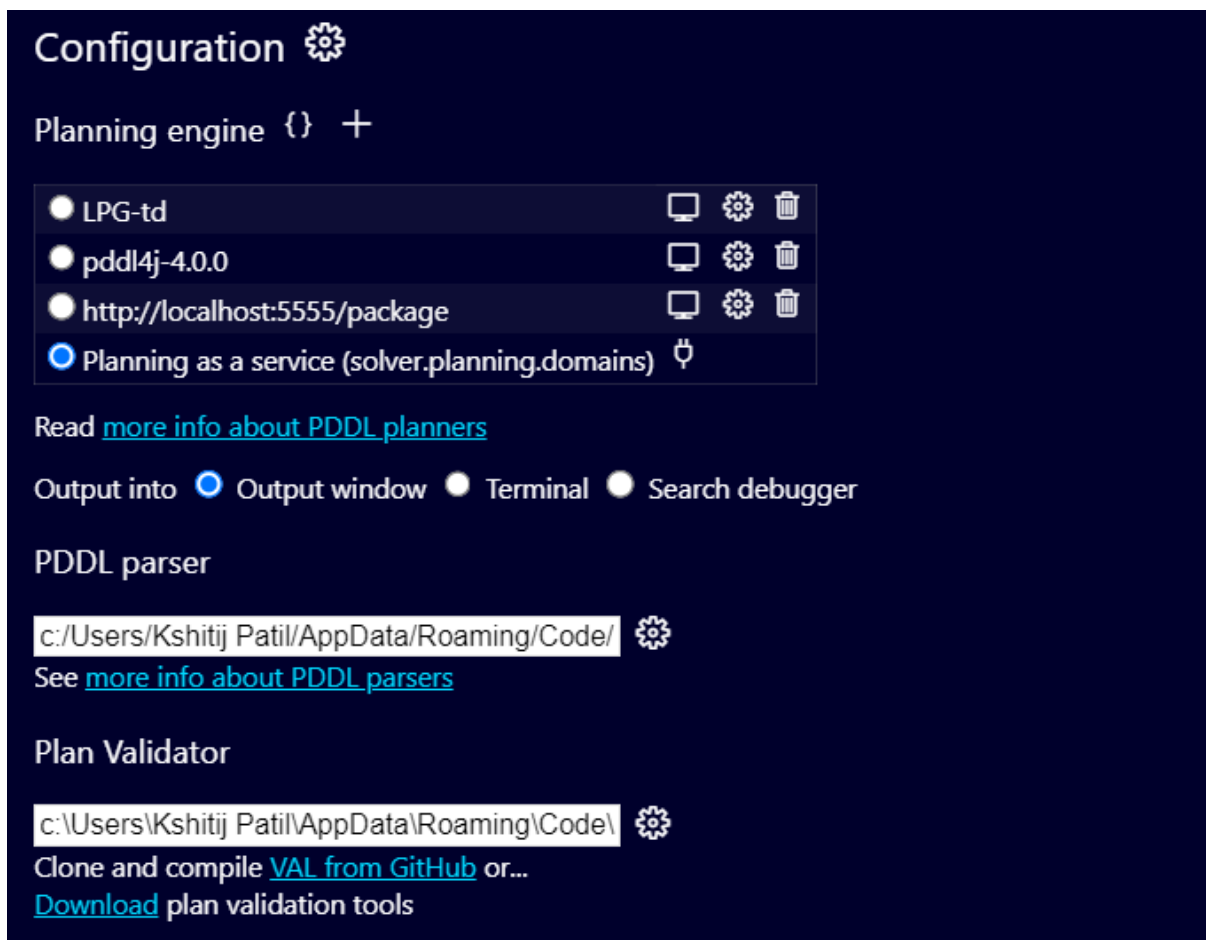
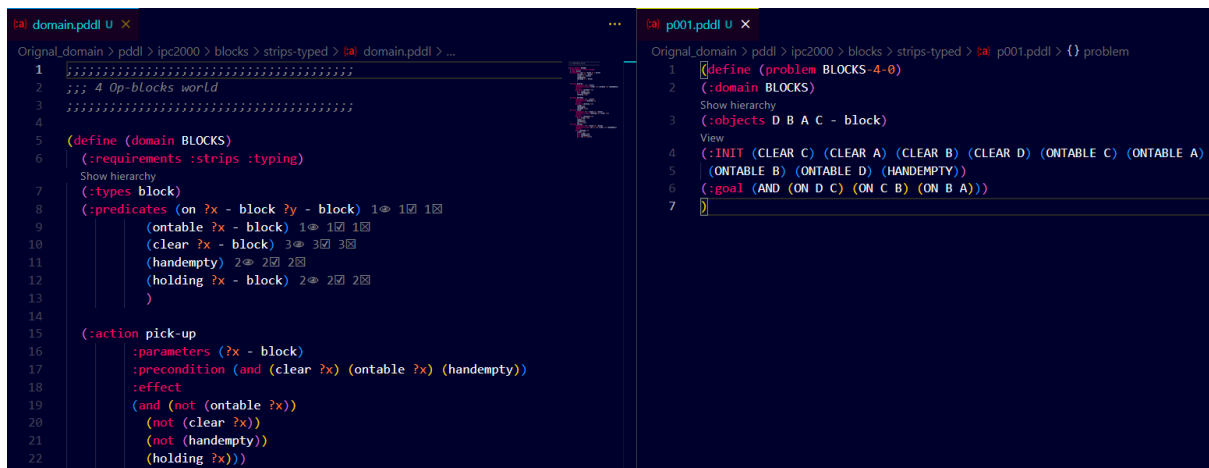


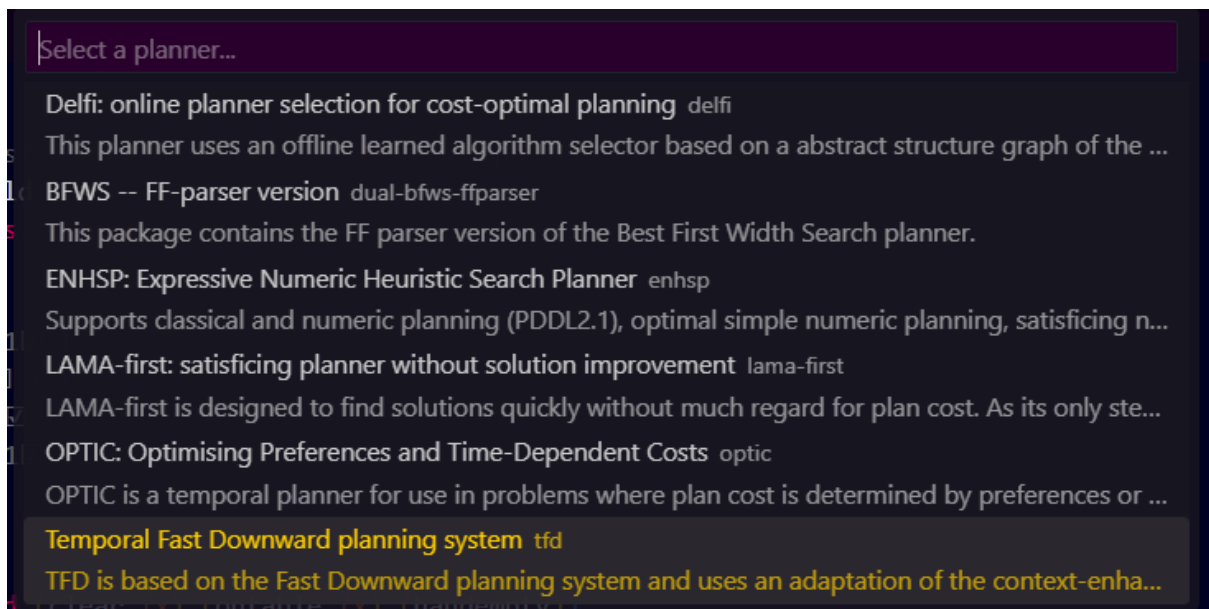
Figure A.1: VS-Code PDDL Overview Page

A Appendix



The image shows two side-by-side Prolog IDE windows. The left window, titled 'domain.pddl U X', displays the content of a domain file. It starts with a comment '4 Op-blocks world' and defines a domain named 'BLOCKS'. The domain includes requirements for 'strips' and 'typing', types for 'block', and several predicates: 'on', 'clear', 'handempty', and 'holding'. An action named 'pick-up' is also defined with its parameters, precondition, and effect. The right window, titled 'p001.pddl U X', shows a problem file. It defines a problem named 'BLOCKS-4-0' within the 'BLOCKS' domain. It lists objects 'D B A C - block' and sets initial conditions for 'INIT' (CLEAR C, CLEAR A, CLEAR B, CLEAR D, ONTABLE C, ONTABLE A) and goal conditions for 'GOAL' (AND (ON D C) (ON C B) (ON B A)).

Figure A.2: Uploading a Domain and its corresponding Problem file



The image shows a terminal window with a dark background and light text. At the top, there is a prompt 'Select a planner...'. Below it, several planner options are listed with their descriptions. The options are: 'Delfi: online planner selection for cost-optimal planning delfi', 'BFWS -- FF-parser version dual-bfws-ffparser', 'ENHSP: Expressive Numeric Heuristic Search Planner enhsp', 'LAMA-first: satisficing planner without solution improvement lama-first', 'OPTIC: Optimising Preferences and Time-Dependent Costs optic', and 'Temporal Fast Downward planning system tfd'. The 'tfd' option is highlighted in yellow. Below the 'tfd' option, there is a brief description: 'TFD is based on the Fast Downward planning system and uses an adaptation of the context-eha...'. The terminal also shows some faint, partially visible text at the bottom, including '(:parameters (?x - block)', '(:precondition (and (clear ?x) (ontable ?x) (handempty))', and '(:effect (and (not (ontable ?x)) (not (clear ?x)) (not (handempty)) (holding ?x)))'.

Figure A.3: Selecting TFD-Planner to generate valid plan

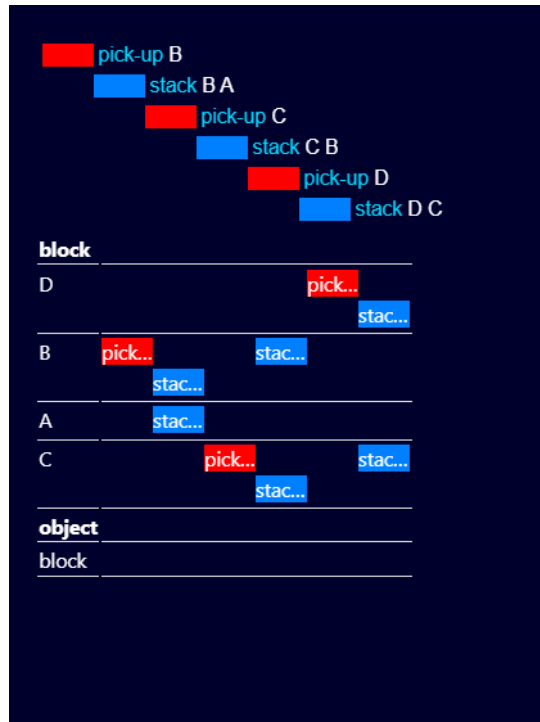


Figure A.4: Generated Plan

```
[t=0.00814414s, 12856 KB] Plan length: 6 step(s).
[t=0.00814414s, 12856 KB] Plan cost: 6
[t=0.00814414s, 12856 KB] Expanded 11 state(s).
[t=0.00814414s, 12856 KB] Reopened 0 state(s).
[t=0.00814414s, 12856 KB] Evaluated 12 state(s).
[t=0.00814414s, 12856 KB] Evaluations: 24
[t=0.00814414s, 12856 KB] Generated 35 state(s).
[t=0.00814414s, 12856 KB] Dead ends: 0 state(s).
[t=0.00814414s, 12856 KB] Number of registered states: 12
[t=0.00814414s, 12856 KB] Int hash set load factor: 12/16 = 0.75
[t=0.00814414s, 12856 KB] Int hash set resizes: 4
[t=0.00814414s, 12856 KB] Search time: 0.000427453s
[t=0.00814414s, 12856 KB] Total time: 0.00814414s
Solution found.
Peak memory: 12856 KB
Remove intermediate file output.sas
search exit code: 0

INFO    Planner time: 0.36s

Plan found:
0.00000: (pick-up b)
0.00100: (stack b a)
0.00200: (pick-up c)
0.00300: (stack c b)
0.00400: (pick-up d)
0.00500: (stack d c)
Metric: 0.005
Makespan: 0.005
States evaluated: undefined
Planner found 1 plan(s) in 2.9secs.
```

Figure A.5: Plan Evaluation using VAL

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature