

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

Master Thesis

An LLM-Modulo Framework for Automated PDDL Domain Model Generation

Srushti Shetye

Course of Study: Computer Science

Examiner: PD Dr. Ilche Georgievski

Supervisor: PD Dr. Ilche Georgievski

Commenced: January 15, 2025

Completed: July 15, 2025

Abstract

Domain model generation is a foundational challenge in automated planning, requiring models that are both syntactically correct and semantically robust for effective plan synthesis and execution. Creating planning domain models in languages such as the Planning Domain Definition Language (PDDL) is time-consuming, error-prone, and requires substantial expertise. This limitation has become a principal bottleneck in scaling AI planning to complex, real-world domains. Recent advances in large language models (LLMs) have showcased their ability to generate structured, code-like representations from natural language descriptions, fueling optimism that these models might automate the transformation of human intent into formal planning models. However, fully automating the generation of domain models from natural language is still a challenge. Outputs from LLMs often have semantic errors, logical faults, and lack the reliability needed for dependable planning. Additionally, current processes frequently need manual adjustments or corrections from experts, which goes against the aim of total automation.

To address these limitations, this thesis proposes a novel feedback-driven framework that enables LLMs to generate PDDL domain models through a fully automated pipeline with no human intervention. The first step is to have the LLM generate intermediate representations called Answer Set Programming (ASP) rules. These ASP rules are then used, again through the LLM, to generate corresponding PDDL domain models via structured prompts and iterative feedback. Semantic coherence and context relevance are ensured through retrieval-augmented generation (RAG), offering PDDL samples to facilitate generation and correction.

Once the domain models are generated, plans are created using the Fast Downward planner and then validated using VAL to ensure logical soundness and successful goal achievement. The framework is evaluated using five advanced LLMs—Gemini 1.5 Flash, Gemini 2.5 Flash, DeepSeek V3, DeepSeek R1, NVIDIA: Llama 3.3 Nemotron Super 49B v1—in the benchmark domains of the International Planning Competition (IPC). Performance is assessed based on accuracy, generation time, makespan, and plan cost, revealing differences in how effectively each LLM adapts to structured feedback. The results demonstrate that LLMs can reliably generate planner-compatible models with few errors through targeted prompting and automated refinement.

Acknowledgement

I express my deep gratitude to my supervisor, PD Dr. Ilche Georgievski, and the Institute of Architecture of Application Systems at the University of Stuttgart for giving me the opportunity to undertake this thesis. From the very beginning, PD Dr. Georgievski's consistent guidance, expert insights, and encouragement were invaluable throughout the duration of this thesis. His constructive feedback and critical perspective greatly shaped the direction and quality of my work.

I would also like to sincerely thank my family for their unwavering support, encouragement, and belief in my abilities. Their constant presence and motivation have given me the strength and also inspired me to stay focused and push through difficult moments.

I hereby declare that generative AI was used exclusively for the revision and editing of self-written text.

Contents

1	Introduction	15
1.1	Overview of the State of the Art	16
1.2	Gaps in the State of Art	17
1.3	Research Question	18
1.4	Proposed Framework	19
1.5	Structure	19
2	Background	21
2.1	AI Planning	21
2.2	Planning Domain Definition Language (PDDL)	22
2.3	Retrieval-Augmented Generation (RAG)	22
2.4	Answer Set Programming (ASP)	23
2.5	Large Language Models (LLMs) and Prompting	23
3	Related Work	25
4	LLM-Modulo Framework	31
4.1	User Input & Chain-of-Thought Prompting	32
4.2	Intermediate Representation via ASP Generation	32
4.3	ASP Verification with External Verifier and Feedback Loop	34
4.4	ASP to PDDL Translation via LLM2 and RAG	35
4.5	Syntactic Validation of PDDL and Iterative Correction	36
4.6	Plan Generation and Semantic Feedback Loop	36
4.7	Semantic Validation of the Generated Plans	36
5	Implementation	39
5.1	Prompt Design	39
5.2	CoT Strategy	40
5.3	Selecting LLMs and API keys	41
5.4	External Verifier and Plan Validator	44
5.5	AI Planner- Fast Downward	45
6	Experimental Evaluation	47
6.1	Benchmarks	47
6.2	Models	47
6.3	Evaluation Metrics	47
6.4	Technical Configuration	51
6.5	Experimental Scenario	52
6.6	Analysis	62

7 Conclusion	69
Bibliography	71
A Appendix	77
A.1 Implementation of Movies98 domain using Gemini 1.5 Flash	77

List of Figures

4.1	Automated Feedback Framework	31
4.2	Intermediate Result (ASP) generation and verification	32
4.3	PDDL generation using RAG, ASP and generating plans	35
5.1	Domain description for Mystery98 Domain	41
5.2	Selection of LLMS	42
6.1	Logic-aware match report for the Satellite domain using DeepSeek V3	51
6.2	Direct success without feedback loops	53
6.3	Success after feedback-based refinement with VAL	54
6.4	Success after feedback-based refinement with Fast-Downward	55
6.5	Success after feedback-based refinement with VAL and Fast-Downward	56
6.6	Fix Attempt 1: Planner fails due to undefined predicate “in_harmony”	57
6.7	Fix Attempt 2: Planner successfully generates plan after feedback-based correction	58
6.8	Errors detected in PDDL files by VAL	59
6.9	Errors detected in PDDL files during Plan generation	60
6.10	Plan generated but failed validation	61
A.1	Movies98 input Natural Language domain description	77
A.2	ASP Generation part 1	78
A.3	ASP generation part 2	79
A.4	PDDL generation started after ASP verified by Clingo	79
A.5	PDDL files verified by external verifiers and generated a plan	80
A.6	Plan validated by VAL	81

List of Tables

6.1	Classification of LLM Output Behaviors	62
6.2	Evaluation of Domain Model Generation using Gemini 1.5 Flash	63
6.3	Evaluation of Domain Model Generation using Gemini 2.5 Flash	64
6.4	Evaluation of Domain Model Generation using Deepseek V3	65
6.5	Evaluation of Domain Model Generation using Deepseek R1	66
6.6	Evaluation of Domain Model Generation using Nemotron Super	67

List of Abbreviations

AI	Artificial Intelligence
CoT	Chain-of-Thought
ASP	Answer Set Programming
RAG	Retrieval-Augmented Generation
IPC	International Planning Competition
LLM	Large Language Model
PDDL	Planning Domain Definition Language
NLP	Natural Language Processing

1 Introduction

Automated planning, also known as AI planning, is a key area of artificial intelligence that enables intelligent agents to autonomously determine sequences of actions to achieve specific goals in complex and dynamic environments [NGT04]. At the heart of these systems is the domain model. This model formally outlines the actions, preconditions, and effects that describe the possible interactions between an agent and its environment [Geo23; MGH+98]. The quality and usefulness of the plans that are produced are directly impacted by the domain model’s expressiveness and accuracy.

The generation of high-quality domain models is seen as one of the biggest challenges in scaling automated planning systems to real-world situations. Writing Planning Domain Definition Language (PDDL) models requires not just defining the structure of actions and predicates but also capturing complex semantic dependencies, causal relationships, and constraints that represent the intended behavior of the domain accurately [Geo23; Pat24; TZM25]. This process is inherently iterative, requiring repeated validation, revision, and often collaboration between domain experts and planning specialists. Constantly updating and verifying domain models can be expensive in contexts which modify frequently. This creates a significant maintenance burden and slows down the use of Artificial Intelligence (AI) planning technologies at scale.

As real-world applications of planning grow to include robotics, industrial automation, logistics, and complex software systems, the challenge of scaling, maintaining, and adjusting these models to meet changing needs has become more pressing. Even small mistakes, such as missing preconditions, unreachable goals, or contradictory effects, can make models unusable. This can lead to plans that fail or are logically flawed [Pat24; SJCG24a]. The development and implementation of planning systems would be simpler for non-experts if the process for generating domain models from natural language were automated. It would also speed up prototyping, help adapt to new fields, and improve resistance to errors in specifications.

More data-driven and AI-supported approaches have replaced traditional, expert-driven workflows in the planning community in recent years. This change has come from the growing complexity of tasks and the need for more flexible, user-friendly planning solutions. As planning areas become more varied and dynamic, often involving multiple agents, uncertain situations, or rapidly changing goals. The responsibility on human modelers to cover all possible scenarios has increased significantly.

Meanwhile, improvements in natural language understanding and machine learning have pushed efforts to connect informal user needs and formal planning representations. With Large Language Models (LLMs) showing a growing ability to process, interpret, and create structured content from text, we have a unique chance to rethink the modeling process. This change could make AI planning more accessible, easier to maintain, and more aligned with user intent. Achieving this level of automation could open up new areas for application and spread the benefits of smart planning across different industries.

Beyond just being a technical convenience, automating the generation of domain models plays a crucial role in making AI planning more accessible to a broader audience while also lightening the cognitive load on users. Many sectors like personalized assistants, smart manufacturing, logistics optimization, and service robotics require regular updates to their planning models to keep up with new tasks, constraints, or user preferences. If there are delays or errors in updating these models, it can result in operational challenges, increased costs, or even safety concerns. By enabling users to express their planning tasks in natural language and converting those requests into accurate domain models, we can significantly speed up innovation cycles and encourage wider adoption of planning technology.

LLMs like GPT-4 and DeepSeek have sparked considerable interest in using their strong natural language processing abilities to connect informal human intent with the formal symbolic representations needed by planners [KVG+24; OSK+24a; VK23]. Moreover, their ability to learn from previous modeling experiences and feedback allows for ongoing enhancements as planning systems adjust to new challenges and requirements. With their extensive training on diverse datasets and emerging reasoning skills, LLMs provide a promising route to streamline domain modeling and encourage innovative human-AI collaboration in the planning field.

The generation of semantically correct domain models from natural language in an completely automated, end-to-end manner is still an open problem. The challenge now lies in converting these capabilities into robust, transparent, and verifiable pipelines that planners can rely on in practice. This thesis bridges the gap by combining LLM-generated models with symbolic validation, improving both the automation and reliability of domain model generation in AI planning.

1.1 Overview of the State of the Art

An increasing number of research focused on the automated generation of planning domain models using LLMs. The goal is to convert structured or semi-structured natural language input into PDDL or similar formats [KVG+24; OSK+24a; SJCG24b; VK23]. Early research demonstrated that, when provided with well-crafted prompts or templates, LLMs could produce legitimate PDDL fragments and even genuine action schemas for traditional planning tasks. Nevertheless, thorough analyses have repeatedly demonstrated that LLM-generated models frequently contain semantic flaws. These include unattainable goals, lacking invariants, and action definitions that are insufficient or inconsistent. The reason for these issues is that LLMs are essentially statistical models of language rather than systems that comprehend logic unique to planning. Because of this, they frequently lack the resources to guarantee action consistency or adhere to domain restrictions [Pat24; SJCG24b; VK23]. To address these shortcomings, recent frameworks such as the Translate-Infer-Compile (TIC) pipeline have introduced intermediate logical representations—most notably Answer Set Programming (ASP)—and incorporated symbolic verifiers like Clingo to check and refine logical consistency prior to PDDL synthesis [ASS24; SJCG24b]. Other lines of work have focused on retrieval-augmented generation (RAG). This framework improves the context of LLM outputs by using validated domain examples. Another area is validation-guided refinement. In this method, external tools give feedback that is regularly included in repeated generation cycles [GVSK23; HLP+23]. Despite these advances, most systems still need manual intervention at various stages, either to correct remaining errors or to curate templates and examples. Ongoing challenges include the lack of integrated, automated correction tools, limited support for complex domain features,

and the absence of closed-loop pipelines that bring together natural language processing, symbolic reasoning, and validator feedback. These gaps show the continued need for strong, scalable methods to fully automate domain model generation, which this thesis aims to address.

The shortcomings of LLMs in handling large type hierarchies, thinking about long-term relationships, and guaranteeing invariants that are essential for efficient planning have been brought to light by recent studies [KVG+24; VK23]. Techniques such as chain-of-thought (CoT) prompting and instruction-tuned models have improved the fluency and coherence of generated outputs. However, issues persist in making sure that generated domain files meet all the syntactic and semantic requirements of downstream planners [OWJ+22]. Some studies suggest incorporating human feedback, interactive learning loops, or curriculum learning to help models understand planning-specific constraints and logical structures [GVSK23; VK23]. Nonetheless, a systematic solution that combines language understanding, knowledge retrieval, formal reasoning, and automated validation in a scalable, end-to-end process is still missing. Bridging these diverse lines of research remains an open challenge and a key focus of this thesis.

1.2 Gaps in the State of Art

The integration of LLM into planning automation has improved, however there are still several significant drawbacks that limit the models' usability and performance. These gaps, which result from flaws in existing methods and technology, point to areas that need more study and advancement.

1.2.1 Limited Error Correction Capability

Current tools like VAL and AI Planners can effectively recognize syntactic errors in PDDL domain files but lack the capabilities to automatically repair these errors. While such tools provide meaningful insights about the kind of errors, manual intervention by experts is normally needed to make corrections. Especially for more intricate semantic errors—where there is an inconsistency between preconditions and effects for an action—there is as yet no robust autonomous system in place that can effectively improve domain models. This is an indicator of the need for hybrid systems in which automatic error detection is complemented by intelligent mechanisms for automatic correction, which might use LLMs with more profound integration in terms of domain knowledge.

1.2.2 Lack of Domain-Specific Context Understanding

GPT-4 and other big language models are capable of providing syntactically valid PDDL domain models, but they frequently lack semantic soundness. This limitation comes from the fact that LLMs, even though exposed to large amounts of data while in training, often lack specialized knowledge to understand complexity and boundaries for individual planning problems. LLMs, therefore, end up producing models that are logically incoherent or violate basic constraints within planning. An example is that they might produce an action with conflicting effects or omit essential predicates to produce defective plans. The study showed that even though optimizing LLMs considerably

reduced syntactic errors, some complex semantic faults were left unresolved. This highlights why domain-specific knowledge needs to be incorporated into LLMs to ensure that generated models fit within the logical structure in the domain.

1.2.3 Absence of Integrated Multi-Stage Feedback Loops Across Planning Phases

Logical reasoning and planning model verification are two phases that require human intervention or are usually managed independently, although they each have their own validation procedures. With present approaches, feedback is frequently applied just locally inside a particular stage, fixing errors without taking into account how they can affect next planning stages. This framework reduces the efficacy of error correction, resulting in duplicate repairs, disjointed iterations, and preventable execution errors in planning. The lack of such comprehensive feedback integration affects the overall quality, coherence, and consistency of automatically generated planning models, especially in LLM-driven systems.

1.3 Research Question

Recent developments in LLMs have demonstrated potential for automating certain steps in traditional AI planning processes. However, the weak domain model generation, limited symbolic reasoning capabilities, and lack of iterative self-correction systems are common problems with present approaches, which often requires human intervention who are domain experts. These issues highlight the need for a structured, feedback-driven approach that incorporates symbolic validators and planning tools. Formal representations such as PDDL domain models will get guidance from this integration as they are developed and refined. To address these issues, this thesis builds upon the LLM-Modulo Framework introduced by [KVG+24] and presents a new modular pipeline that integrates intermediate ASP encodings, external symbolic verifiers (Clingo, VAL), and AI planners—augmented with an iterative self-correction mechanism. The goal of this framework is to make it possible to automatically generate reliable and significant planning domain models.

The research questions aim to fill the existing gaps by exploring how iterative refinement, external validation, and contextual memory affect LLM-based planning model generation. The findings will shed light on how modular LLM frameworks can progress from one-time generation to more reliable and verifiable planning domain generation. This main question aims to understand how a system that combines external verifiers, feedback backprompted can create high-quality, validated planning models from natural language input.

With this goal in mind, the research looks into the following main question:

- **RQ1:** How can an LLM-Modulo framework allow for the independent and verifiable generation of PDDL domain models for automated planning tasks?

To explore this central question, the study addresses the following sub-questions:

- **RQ1.1:** How effective are feedback loops, such as backprompting with Clingo and VAL outputs, in refining intermediate ASP and final PDDL outputs?

This sub-question looks at the iterative correction process in the pipeline. It examines if feedback from external verifiers significantly helps improve the accuracy during different phases of generation.

- **RQ1.2:** How do symbolic validation tools, like Clingo for ASP and VAL and Fast Downward for PDDL, affect the correctness and stability of generated planning models?
- **RQ1.3:** How do different LLMs perform regarding convergence rate, output validity, and the number of feedback iterations needed across various International Planning Competition (IPC) domains? This question looks at whether this modulo optimized for logical reasoning LLMs works better or faster than general-purpose LLMs when generating domain models.

1.4 Proposed Framework

To address the research questions, this thesis introduces a fully automated and modular framework that combines LLMs with symbolic validation and iterative feedback. The central goal is to move beyond one-shot generation by enabling a process that can autonomously produce and refine planning domain models from natural language input—without requiring expert intervention. The framework is designed to ensure that the outputs are not only syntactically correct but also semantically meaningful within the planning context.

This proposed direction directly responds to the overarching research question by establishing a structure where each component—generation, validation, and refinement—contributes to the generation of reliable PDDL models. Feedback mechanisms address RQ1.1 by enabling correction through verifier outputs. Embedded symbolic tools support RQ1.2 by enforcing correctness and logical consistency. Comparative testing across different LLMs helps answer RQ1.3, revealing how model performance varies in terms of output quality and refinement needs. The full technical architecture is detailed in the Chapter 4, but this section outlines the thesis’s commitment to building an end-to-end, verifiable pipeline for automated domain modeling.

1.5 Structure

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

- **Chapter 2:** An detailed background of AI planning, ASP, PDDL, LLM prompt approaches (e.g., CoT), and tools (e.g., Clingo, Fast Downward, and VAL) is introduced in this chapter. For understanding the technological tools used throughout the pipeline, it provides the fundamental knowledge required.

- **Chapter 3:** This chapter critically reviews state-of-the-art literature on LLM-based domain modeling, planning, and validation. It compares various neuro-symbolic frameworks and feedback-based systems, identifying their strengths and limitations. It also positions the proposed work in the broader research landscape.
- **Chapter 4:** This chapter presents the architecture of the proposed system. It describes the modular, multi-phase pipeline, including preparation, domain modeling, and model assurance. Focusing on the iterative backprompting loops and the role of external validators such as Clingo and VAL in producing planner-ready domain models from natural language input.
- **Chapter 5:** The technical implementation of the suggested framework is covered in detail in this chapter. It describes how different stages, such as the planners, ASP generators, verifiers, and LLMs, are related to one another and how each module adds to the refinement loop.
- **Chapter 6:** This chapter describes the experimental setup, hardware and software platform, LLM comparison strategy, and assessment metrics. The performance is examined based on plan generation status, makespan, plan cost, total generation time, number of required feedback loops, and accuracy.

2 Background

A thorough summary of the fundamental elements supporting this thesis is given in this section. These elements include AI planning, PDDL, ASP, LLMs, and the assessment tools. Each subsection contextualizes the methods and tools used in our proposed framework, drawing upon established literature to highlight their significance and relevance.

2.1 AI Planning

AI Planning, or Automated Planning, is a discipline in artificial intelligence focused on generating sequences of actions called plans, that transition an agent or system from an initial state to a goal state. It originated in the mid-1960s from the need to provide robots with autonomous decision-making abilities and has since evolved into a vibrant field intersecting with areas like scheduling. Most approaches to planning model the world as a state transition system: a set of states, a set of available actions, a defined initial state, and a set of goal states. The planning task is then cast as a search for a path through this state space from the initial state to one that satisfies the goals, using only allowed actions [AG24].

A state model in AI planning consists of the following:

- a finite set of states,
- an initial state,
- a set of goal states,
- a finite set of actions, and
- a deterministic transition function mapping actions and states to successor states.

Classical planning operates under assumptions of a static, deterministic, and fully observable world. It typically models planning problems with discrete time, finite state spaces, and deterministic outcomes Silver et al. [SATK22]. The complexity of planning arises because the number of possible states and actions grows rapidly with domain size, making the search for valid plans computationally challenging. The goal is to derive optimal or feasible plans that satisfy all preconditions and achieve the defined goal states. While this has worked effectively in constrained settings, real-world domains increasingly require scalable solutions that minimize the need for hand-coded models. As planning problems grow in complexity, the construction of accurate and complete domain models has become one of the main bottlenecks in the deployment of AI planning systems [VCM+15].

2.2 Planning Domain Definition Language (PDDL)

The PDDL, which was first presented by [MGH+98], is the most widely used modeling language in classical planning. It standardizes the representation of actions, preconditions, effects, and domain constraints. PDDL is a domain-specific language developed to standardize the way planning problems are described [MGH+98]. It allows for defining two core files:

- **Domain File (domain.pddl):** Describes types, predicates, and actions, each defined with their respective preconditions and effects.
- **Problem File (problem.pddl):** Specifies objects, initial state, and the goal state to be achieved.

With its Lisp-like syntax of nested parentheses, PDDL is predicated on first-order logic. Its formal structure is based on relations and objects: actions define possible transitions between states by defining preconditions and effects, while predicates explain facts about objects [AG24].

Logical connectives like not, and, or, forall (universal quantification), and existing (existential quantification) can be used in PDDL to characterize complex circumstances. The division of domain and problem descriptions facilitates benchmarking and modularity by enabling the reuse of a single domain definition across several problem instances. With each iteration, PDDL has grown to incorporate more complex features such as temporal actions, derived predicates, conditional effects, constraints, plan preferences, object typing, and numeric fluents.

PDDL's formal syntax enables planners to reason over actions and state transitions systematically. It supports a wide array of planning constructs including conditional effects, numeric fluents, temporal actions, and object typing. However, writing syntactically and semantically correct PDDL models remains a challenging task, especially for non-experts [OSK+24a; SATK22].

2.3 Retrieval-Augmented Generation (RAG)

RAG is a hybrid setup that improves the factual accuracy and performance of LLMs by combining generation with information retrieval. Traditional LLMs often have problems with outdated or inaccurate outputs because of limits in their training data. RAG solves this issue by allowing the model to search an external knowledge source, such as a database, corpus, or document store, during inference. It then integrates the retrieved information into its responses. The setup usually includes two main parts: a retriever that finds relevant documents based on the input query, and a generator that bases its output on both the query and the retrieved documents [HH24; LPP+20]. This setup closes the gap between static pretraining and dynamic knowledge access. It allows LLMs to produce more accurate and current outputs.

In planning contexts, RAG is key in grounding model outputs, such as PDDL domain models, using valid, reusable prior examples. This is especially helpful when models generate structured outputs that need to follow strict syntactic and semantic rules. By using a retrieval corpus, such as standard IPC planning domains, the generation process is informed by structurally correct samples. This improves both correctness and interpretability [HH24]. RAG is not just a way to improve knowledge; it is also a tool that helps match LLM outputs with expert-designed domain schemas. This is important for planning pipelines, like the one discussed in this thesis.

2.4 Answer Set Programming (ASP)

A declarative programming approach called ASP is designed to solve challenging search and reasoning issues [GKKS14; Lif99]. Using logic rules, facts, and constraints, ASP enables users to model issues. It then depends on solvers to provide stable models (also known as answer sets) that meet the requirements. ASP is particularly powerful for nonmonotonic reasoning, where new information can invalidate previous conclusions [BET11].

ASP offers a concise and expressive method for encoding action preconditions, transition effects, and domain dynamics in the planning context. It makes it possible to model conditional effects, frame axioms, and inertia in a logical and natural way. Lifschitz demonstrated how ASP can be used to encode classical planning tasks by mapping states, actions, and goals into logic programs [Lif99]. The resulting answer sets correspond to valid plans. ASP's structured logic makes it ideal for integration with symbolic verifiers like Clingo, which can identify structural and syntactic inconsistencies.

For instance, consider a simple domain with locations A and B and an agent needing to move between them. In ASP, one might encode this as:

- **Facts:** `location(a). location(b). at(a, 0).`
- **Actions:** `action(move(X,Y)) :- location(X), location(Y), X != Y.`
- **Transition Effects:** `at(Y, T+1) :- occurs(move(X,Y), T), at(X, T).`
- **Frame Axioms:** `at(L, T+1) :- at(L, T), not -at(L, T+1).`
- **Goal:** `:- not at(b, maxTime).`

This formulation supports rigorous logical reasoning and is a natural fit for intermediate representation in our pipeline.

2.5 Large Language Models (LLMs) and Prompting

LLMs constitute a revolutionary innovation in artificial intelligence to the extent that machines can comprehend as well as produce human language with unprecedented expressiveness as well as flexibility. Composed largely using the Transformer architecture, LLMs are distinguished by their vast scale, with tens or even hundreds of billions of parameters in the case of typical LLMs. They are trained on huge corpora of text [WTB+22]. The history of language modeling has unfolded from statistical models, including the use of n-gram strategies, then neural language models using recurrent neural networks, to recent moves towards pre-trained language models (PLMs) using large-scale unsupervised or self-supervised learning [ZZL+25].

LLMs work through the generation of the probability distribution over word sequences, usually through the use of next-token predictability. Training the models is computationally demanding, with the use of distributed optimization techniques, as well as advanced engineering to deal with the size of the models as well as the data. With the growth in the size of the models, researchers

2 Background

have observed the 'scaling law' where enhanced model capability as well as better downstream task performance is bound to follow a definite trend with regard to model parameters, data size, as well as computational effort.

A defining feature of LLMs is the emergence of novel abilities that are absent in smaller models. These emergent abilities such as in-context learning, instruction following, and multi-step reasoning—manifest abruptly when the model surpasses a certain scale, rather than improving incrementally with size. This phenomenon is akin to a phase transition, where qualitative changes in model behavior arise from quantitative increases in scale. Take few-shot learning, for example: LLMs learn to tackle new tasks straight from the examples provided in the input prompt, without any new parameters being fitted. Similarly, with some smart prompting (for instance, with a CoT prompt), the resources of larger models can be organized to answer complex questions that are quite beyond smaller ones.

A particularly effective prompting technique is CoT prompting, which encourages the LLM to decompose reasoning tasks into step-by-step logical fragments [WWS+22]. This is especially useful when generating structured outputs like ASP or PDDL, as it forces the model to internalize the logical dependencies involved. However, despite their fluency, LLMs lack the formal reasoning abilities to ensure logical soundness. This makes them ideal candidates for integration with external symbolic verifiers and refinement loops. LLMs have transformed from how Natural Language Processing (NLP) is approached, shifting from custom models, to general, adaptable systems that can be to a wide range of natural tasks and problems. Their impact can be seen in the wide array of problem to which they can be applied.

3 Related Work

Learning action and domain models has been a central challenge in the automated planning community for many years. The process of manually creating these models using formalisms like PDDL can be quite tedious and often leads to errors, which has motivated researchers to find ways to automatically generate them from data. Early attempts mainly focused on using fully or partially observed plan traces, demonstrations, or state transitions to deduce preconditions, effects, and action structures. Several surveys have classified these approaches according to their data sources, levels of supervision, and the types of models they target. A significant contribution in this field is the work by [AJO19], which presents the learning of STRIPS action models as a classical planning problem. Their compilation-based strategy revealed that existing planners could be utilized to address model learning tasks using logical inference alone. The use of natural language models, especially LLMs, to generate domain models from natural language descriptions has gained attention. This section examines the main contributions made by more recent LLM-based frameworks for domain acquisition planning as well as traditional learning paradigms.

The research article “A Comprehensive Framework for Learning Declarative Action Models” offers a detailed framework for understanding how declarative action models, which are compact representations of system state transitions, are learned in AI planning. The authors define the learning process using state constraints and combinatorial search. They also categorize prior work by target models, learning data, guidance functions, and learning algorithms. Their framework stands out for linking model learning to classic AI problem-solving methods and for examining the strengths and weaknesses of current approaches. The study highlights that in order to develop generalizable models, learning frequently depends on having access to observations or plan execution traces. The success and data dependencies of traditional domain model learning are highlighted in this analysis, which supports our thesis by motivating us to investigate natural language-driven methods independent of such empirical data [AJO22].

The survey “A Review of Learning Planning Action Models” systematically examines the landscape of techniques for acquiring planning action models from data. From completely observed plan traces to partial state observations, the taxonomy of learning approaches differentiates them based on the types of observational input and the degree of supervision needed. This research reviews different strategies for managing incomplete or noisy data and discusses the challenges that come from needing high-quality empirical traces. A significant contribution of this review is its critical look at the field’s dependence on trace-based methods and its identification of difficulties in applying findings across areas with limited observational data. This research is pertinent to our thesis as it points out the challenges of data-focused learning and highlights the possible benefit of using alternative sources like natural language descriptions [AFP+18].

The survey “A Review of Machine Learning for Automated Planning” offers a detailed look at how machine learning methods are used to automate the generation of planning action models and to manage control knowledge. The authors focus their review on two main issues: learning action

3 Related Work

models and learning search control. They also discuss reinforcement learning approaches. The survey points out the main strengths and weaknesses of each method. It stresses that successful learning relies significantly on having access to rich plan traces, execution examples, or user-generated data. This paper is pertinent to our thesis due to the cohesive approach to machine learning for planning. It demonstrates how existing approaches rely too heavily on empirical data and lays the groundwork for investigating approaches that employ natural language as the primary input for model learning [JDF+12].

The article “LLMs Can’t Plan, But Can Help Planning in LLM-Modulo Frameworks” offers a critical look at how LLMs are used in automated planning. The authors argue that although LLMs are great at producing clear and coherent text, they do not have the logical strength required for plan generation and self-checking. This is shown by thorough tests on top models like GPT-4, Gemini, and Claude. The authors demonstrate that when LLMs are used in an independent environment, they rarely generate correct, executable plans. They also struggle to effectively critique and improve their own outputs, even when using advanced prompting techniques like CoT or iterative self-critique. To address these limitations, the paper presents the LLM-Modulo framework. This framework treats LLMs as useful, approximate knowledge sources within a well-integrated neuro-symbolic pipeline. In this framework, LLMs generate candidate plans, domain models, or knowledge pieces. However, all outputs must undergo formal verification by external model-based critics, like automated planners or validators, before they are accepted. This bi-directional, generate-test-critique loop allows LLMs to support planning and reasoning tasks in collaboration with sound symbolic systems, leveraging their generative capabilities without overestimating their logical competence. Human involvement is especially important in the knowledge acquisition and model specification phases. This highlights that the approach is not a completely automated solution. This research is relevant to our thesis because it shows the promise and need for combining LLMs with external formal validators to ensure correctness. Our work is based on this modular idea. We use intermediate representations, like ASP, multi-stage formal validation with Clingo and VAL, and iterative feedback methods. However, we go beyond just plan generation to cover the entire process of domain model induction and repair [KVG+24].

The paper “Large Language Models as Planning Domain Generators” by explores how to automate the generation of PDDL domain models using LLMs. The authors provide an evaluation framework that uses LLMs to convert natural language descriptions into PDDL. They measure the results with automated syntactic and plan-level tests, including Action Reconstruction Error and heuristic domain equivalence. This method is an improvement over previous studies that relied on subjective human assessment. It offers a scalable and objective way to evaluate model quality. Their research tests seven LLMs across nine planning domains. The results indicate that although larger models can identify key aspects of domain models, their outputs frequently have ongoing syntactic errors and semantic mismatches. Even correct PDDL syntax does not ensure similar planning performance. This points out a gap between possible generation and real usability. The authors conclude that better feedback and correction methods are necessary for strong domain model induction. These insights directly support our thesis, which tackles these issues by integrating formal validation and feedback-driven improvement into the LLM generation process. This approach brings us closer to practical and dependable automation in planning domain engineering [OSK+24b].

The article “Generating Consistent PDDL Domains with Large Language Models” looks at generating executable PDDL domains from natural language input using LLMs. The authors propose a new pipeline that combines structured prompting, automated consistency checks, and

reachability analysis. It introduces a feedback loop in which any discrepancies are iteratively fixed by the LLM. By incorporating error identification and repair, this approach reduces the amount of expert intervention required for domain model validation, improving upon previous LLM-based approaches that produced code or fragments. Their experiments in standard and custom planning domains show that the feedback-driven pipeline cuts down on errors and reduces the need for manual correction. However, limitations still exist. Full semantic correctness is not always reached, especially in more complex domains, and some human verification is still required for difficult cases. The results emphasize the value of automated validation and feedback, but they also point to the need for further advancements to handle complicated planning scenarios. In order to expand on this, our thesis integrates multi-stage formal validators into the automated feedback loop for refinement procedure [al24b].

The thesis “LLMs for AI Planning: A Study on Error Detection and Correction in PDDL Domain Models” explores how well LLMs, especially GPT-4, can find and fix errors in PDDL domain models. The study looks at how effectively LLMs detect and correct both syntax and meaning problems. These issues vary from missing parentheses to complex logical dependencies. The results show that while LLMs can reduce some types of errors and slightly improve model accuracy, they have difficulty with deeper domain-specific semantic inconsistencies and complex logical interdependencies. Often, this requires manual supervision and expert help to achieve full refinement. This work shows that LLM-based methods are promising for automating basic error detection and correction. However, they still do not fully achieve independent domain model improvement. Our thesis implements a validator-guided, closed-loop pipeline that uses ASP and Clingo for structural checks and VAL for semantic validation. By systematically feeding back detailed validator error messages into the LLM for iterative improvement, our approach does not involve any human intervention. This moves us toward a more fully automated framework for domain model generation [Pat24].

The survey “Understanding the Planning of LLM Agents: A Survey” gives a thorough review of new strategies for LLM-based planning. It categorizes the field into task decomposition, multi-plan selection, planner-aided reasoning, self-adjustment and refinement, and memory-enhanced planning. The authors point out the growing use of hybrid neuro-symbolic architectures. They also identify a key issue in developing fully autonomous planning systems. This issue is the lack of mature solutions for generating automated domain models and for robust, validator-driven correction. However, since the survey is mostly theoretical, it does not put these techniques into practice or test them, especially for domain model synthesis. To address these gaps, our thesis puts several of the survey’s proposed directions into action within a unified, fully automated pipeline. We use ASP-based intermediates for effective task breakdown, include iterative planner feedback through Clingo and VAL for semantic and structural improvement, and apply retrieval-augmented prompt engineering to steer self-improvement of LLM outputs. By combining these methods into a closed feedback loop, our work moves from theory to practice, providing a complete automated solution for generating and validating domain models. This realizes the vision expressed in the survey and tackles its noted limitations [HLC+24].

The paper “Translate-Infer-Compile (TIC) for Accurate Text-to-Plan Using LLMs and Logical Representations” by Agarwal and Sreepathy introduces the TIC method as a framework for converting natural language task descriptions into PDDL models. The approach has three main stages. First, it translates natural language into intermediate logical rules, which are often specified or refined by domain experts. Second, it uses logical inference to expand or complete the model.

3 Related Work

Finally, it compiles the results into valid PDDL syntax. This process effectively reduces translation errors and connects language with planning formalism. However, it relies heavily on human involvement, especially for creating and reviewing intermediate rules and correcting outputs that cannot be fully automated. TIC does not include step-by-step validation or correction with outside validators after PDDL generation. As a result, logical or semantic errors can remain after the initial compilation. TIC is important to our work because it clearly shows the value of using intermediate logic layers and modular design in text-to-plan systems. Our thesis builds on this by adding validator-driven checkpoints and automated feedback loops. This approach seeks to achieve ongoing, validator-guided improvement while reducing human involvement in generating the domain model [al24a].

The article “VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL” introduces the VAL tool as a standard for validating PDDL plans. VAL offers strong support for PDDL features like continuous effects and allows for mixed-initiative planning cycles. In these cycles, humans and automated systems work together to generate and refine plans. While VAL plays a key role in making sure plans are semantically correct and executable, its design supports workflows in which humans construct or revise plans with guidance from VAL. The main innovation of VAL is its semantic depth and ability to manage complex planning features in the real world. However, it mainly serves as a post-hoc validator instead of an automated part of the domain model generation process. Our thesis builds on VAL’s strong feedback mechanisms by integrating its semantic validation directly into the model-building pipeline. We adapt and expand the use of VAL from a post-hoc plan checker to an active participant in model development, using its semantic feedback not only to ensure plan soundness but to identify and resolve domain modeling errors throughout the construction process [HLF04].

The paper “There and Back Again: Extracting Formal Domains for Controllable Neurosymbolic Story Authoring” discusses a neurosymbolic method for narrative planning. It uses language models to pull formal, PDDL-style domain representations from natural language stories. The paper also describes how narrative planners help generate structured stories. Their system, Recompose, acts as a human-in-the-loop tool. Experts or authors need to review, revise, and correct both the logical story specifications and the outputs produced by the system. Validation and debugging are based on feedback from the planner. This ensures creative control and model correctness but maintains manual oversight at key stages. The study demonstrates the advantages and difficulties of integrating LLM expressiveness with formal logic and planner-based validation, despite its concentration on creative story domains. Our thesis is relevant because it shows how planners and LLMs can be used together for domain modeling in an efficient manner. It also shows how validator-driven, automated refinement cycles can be used to overcome human-in-the-loop workflows, which encourages both rigorous domain engineering and narrative creativity [al23].

The paper “Leveraging Pre-trained Large Language Models to Construct and Use World Models for Model-based Task Planning” by introduces a hybrid workflow. In this setup, pre-trained LLMs generate PDDL domain models that are then improved through a cycle of feedback from humans and validators. Here, LLMs serve as helpers in planning by building symbolic world models instead of acting as complete solutions. Human experts play a key role; they check validator error messages and offer more corrections. This ensures that complex domain knowledge is captured accurately and that significant errors are fixed. The system shows better plan success rates compared to basic LLM outputs, especially due to the iterative revisions guided by human input and validators. However, relying on manual review for each validation cycle is still a major bottleneck. The

direct mapping from LLM output to PDDL also lacks an intermediate formalism for localized reasoning or better error diagnosis. In our work, we use this paradigm as a motivating example of validator-assisted domain model construction. We specifically tackle its scalability and granularity issues by introducing an ASP-based intermediate layer. This change allows for more precise localization and automated error correction. It enables better feedback cycles driven by validators and lessens the need for repeated expert intervention [GVSK23].

4 LLM-Modulo Framework

[KVG+24] introduced the LLM-Modulo Framework where they highlighted that LLMs are powerful but have important limitations as generative agents in automated planning. Their work shows that while LLMs cannot produce or check plans independently with logical precision, we can use their creative capabilities within a hybrid neuro-symbolic pipeline. In this system, external critics and validators help guide, refine, and correct the model's outputs upto 3 iterations by each external verifier.

Building on this idea, we propose a new LLM-Modulo Framework that adds multi-stage validator-driven feedback loops. We include ASP for intermediate reasoning and RAG to provide the LLMs with relevant domain knowledge and examples at each step. This method allows for strong, end-to-end automation of domain model generation, reducing the need for human help and improving both syntactic and semantic accuracy.

The pipeline operates as follows:

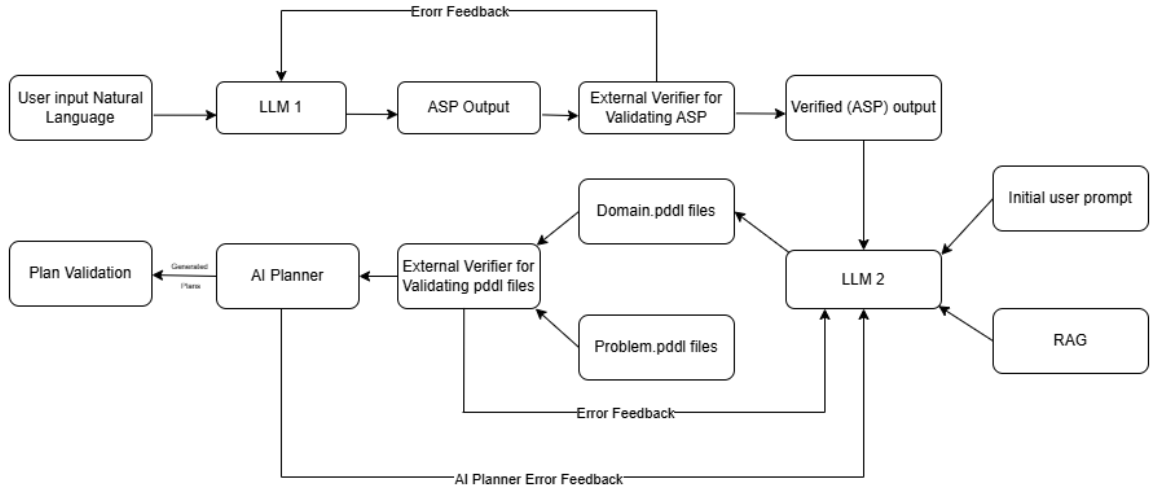


Figure 4.1: Automated Feedback Framework

High-Level Framework Overview

The proposed framework aims to automate the creation of strong, planner-ready PDDL domain models from natural language task descriptions as seen in Figure 1. It builds on the LLM-Modulo Framework by including intermediate representations ASP, multi-stage formal validation, and RAG for context grounding. The overall process (see Figure 4.1).

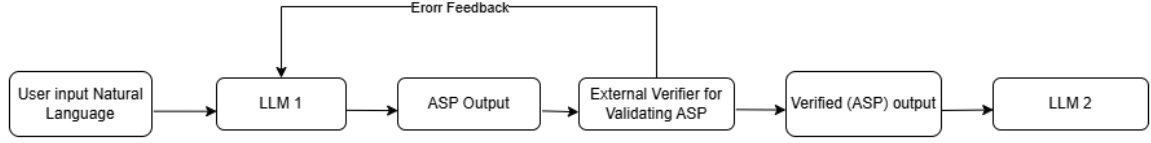


Figure 4.2: Intermediate Result (ASP) generation and verification

For Steps 4.1 to 4.3 refer Figure 4.2

4.1 User Input & Chain-of-Thought Prompting

In our thesis, we use domain-specific natural language descriptions, called encodings. These were introduced in the paper “Automating the Generation of Prompts for LLM-based Action Choice in PDDL Planning” [SFHK25]. The authors present a detailed method for generating these encodings from symbolic PDDL domain files with a GPT-4. The process starts by extracting structured components from the PDDL files. This includes the list of domain actions, their preconditions, and their effects. These symbolic elements are then turned into natural language statements using predefined templates and rules. This produces descriptions that are both meaningful and grammatically correct. The LLM refines this intermediate representation by rewriting the templated content into fluent, easy-to-read narratives. Each encoding outlines: (i) the actions available in the domain, (ii) the logical constraints for each action, and (iii) the state changes caused by those actions. The authors clarify that these prompts were generated through their automated process, not written by hand. To ensure quality, they conducted human evaluations. They showed that the LLM-generated encodings are comparable in structure, precision, and task relevance to human-written ones, making them suitable for tasks like natural language-based action selection and symbolic planning. In our work, we use these verified prompts as high-quality natural language inputs for our own symbolic model generation process.

4.2 Intermediate Representation via ASP Generation

The initial CoT is provided to LLM1, which generates an intermediate result format. Translating natural language descriptions directly into symbolic domain models presents several challenges—particularly in planning tasks where precision, logical consistency, and semantic validity are critical. LLMs frequently fail to meet these structural requirements, despite their ability to produce fluent text. To address this limitation, we introduced an intermediate representation between the natural language input and the final PDDL output. The following subsections outline the necessity of such an IR and the reasoning behind the chosen formalism used to instantiate it.

4.2.1 Why Intermediate Result

End-to-end direct translation of natural language into executable PDDL domain models with LLMs has proven to be unreliable because of inherent limitations in symbolic reasoning and structural fidelity. [OSK+24a; SATK22] show that the frequent output of such models is syntactically acceptable but semantically incomplete domain specifications, with high rates of ill-formed predicates, inconsistent effects of actions, and absent typing constraints. [SJCG24a] add that while such models are structurally possible, they tend to fail under formal verification, which reflects an enduring disparity between linguistic fluency and logical correctness.

This shortfall derives from the token-level, distributional character of LLMs, which have no explicit representations of causal relations and domain-specific constraints essential in symbolic planning. As explained in [KVG+24], LLMs are not independent planners but need to be embedded in mixed neuro-symbolic architectures that unite generative power with symbolic verification.

To solve this, the suggested pipeline places an explicit intermediate representation, in-between the input natural language and the output PDDL model. This decoupling at the architecture supports staged reasoning and verification, where incremental imposition of syntactic and semantic constraints is performed. The use of IR in the design not only obviates the semantic inconsistencies seen in direct LLM-to-PDDL pipelines but also offers formal support for feedback refinement procedures.

4.2.2 Selection of Intermediate result

Numerous intermediate formats have been proposed in the earlier research to organize planning knowledge prior to PDDL translation. These include domain-specific logic templates [AF18], XML/JSON abstractions like itSIMPLE [VTS13], hierarchical task networks (HTN) [GA15]. However, each of these formalisms has limitations in terms of expressiveness, tool integration, or formal verifiability.

By contrast, ASP presents an excellent tradeoff among logical expressiveness, solver support, and translatability. [HDN12] effectively utilized ASP in robotic navigation with semantic constraints, confirming the potential of ASP in reasoning-intensive applications. It was also noted by [JYW+18] that representations based on ASP performed well in logistic applications that included complex precondition modelings and default reasoning. Additionally, ASP allows for dynamic validation and refinement because it is semantically meaningful and compatible with external verifiers like Clingo, unlike XML/JSON formats.

ASP is, therefore, utilized in this work as the preferred intermediate representation because of its representational power, certifiability, and multi-stage neuro-symbolic pipeline compatibility. Its use supports layered verification, error identification at an early stage, and traceability from intent to executable planning logic—enabling improved robustness and correctness in domain models produced by LLM.

ASP, which is a declarative logic formalism with established semantics [Lif99], can support rule-based representation of preconditions, fluents, and effects and is easily verifiable via ASP solvers like Clingo [GKKS19]. In addition, the use of ASP is consistent with more recent developments in neuro-symbolic planning, particularly the efforts of [HDN12] and [JYW+18],

where representations at an intermediate logic stage improve interpretability, error localization, and future translatability. The use of IR in the design not only obviates the semantic inconsistencies seen in direct LLM-to-PDDL pipelines but also offers formal support for feedback refinement procedures.

Incorporating ASP as an IR, then, plays both functional and methodological roles: it decreases the cognitive load upon LLMs, imposes symbolic coherence before PDDL generation, and facilitates iterative refinement through programmatic verification—as all are necessary to produce runnable, planner-compliant domain models.

4.3 ASP Verification with External Verifier and Feedback Loop

Once the intermediate ASP representation has been generated, the pipeline proceeds to a crucial validation phase that ensures its syntactic correctness before moving on to downstream translation stages. This validation is performed using an external verifier, a dedicated tool designed specifically to parse ASP programs and analyze their structural integrity. Acting as a standalone checkpoint within the broader system, this verifier is responsible for identifying a variety of common syntactic issues. These include, but are not limited to, improperly grounded rules, the presence of unsafe variables, references to undefined atoms, and other malformed or ill-formed rule structures that could compromise the logic program’s validity.

This stage showcases widely accepted best practices in developing neuro-symbolic systems, where early formal verification is crucial for ensuring the integrity of the entire process. By identifying structural issues early on, the system can prevent the risk of cascading errors that might carry over into later stages, making them tougher to spot and fix. When the verifier detects a syntactic issue, it creates detailed and structured diagnostics that clearly explain both the type of violation and its location in the program. These diagnostics aren’t just noted down—they’re actively turned into feedback prompts that are sent back to the LLM, providing it with specific guidance for making necessary revisions.

This feedback mechanism is not just open-ended; it operates inside a preset repeating cycle. The LLM is given a set number of opportunities to refine and correct the ASP code in response to the verifier’s errors. The method focuses on improving the code in every cycle by resolving the specific issues that the verifier found. The code is sent back for validation after revision, and this cycle is repeated until the code either satisfies the maximum number of permitted iterations or passes all syntactic checks. The most recent iteration of the ASP representation, however maybe prone to errors, is saved and proceeded to the subsequent step in the pipeline. By combining automated syntax checks with a step-by-step refinement process, this framework guarantees that well-structured logic programs make it through the pipeline. This method enhances the structural integrity and overall reliability of the system, reducing the likelihood of failures later on and enhancing the robustness of the domain modeling process as a whole.

4.4 ASP to PDDL Translation via LLM2 and RAG

Once the ASP model has been verified for syntactic correctness, it is passed to a LLM2 for generating PDDL domain models. This generation process leverages multiple sources of information:

- **Original Natural Language Input:** Retaining the original user scenario ensures that high-level goals and domain intent are preserved, reducing the risk of semantic drift in pddl generation.
- **Verified ASP:** The syntactically valid ASP serves as an intermediate logic-based representation, offering a modular and error-localizable foundation for PDDL generation [ASS24; GVSK23].
- **RAG:** Contextual examples are dynamically retrieved from a curated database of planning models—such as the IPC PDDL corpora—using RAG methods. RAG was used in this pipeline to tackle the known limitations of LLMS in generating logically consistent and domain-specific structured outputs. While LLMs excel at generating fluent text, they often invent functions, types, or predicates in formal languages like PDDL, especially when trained on general data without specific domain knowledge [KVG+24; OSK+24a]. RAG mitigates this problem by using dependable PDDL corpora, like IPC planning domains, to guide the generation process. During generation, this retrieval step guarantees that the model can access authentic syntactic structures and significant planning notions. Consequently, it raises the likelihood of generating outputs that complement planners. Empirical studies in the neuro-symbolic planning literature (e.g., [ASS24; SFHK25]) show that adding RAG to code or domain model synthesis pipelines boosts structural fidelity and success rates in plan generation. RAG provides LLM2 with concrete, high-quality domain model templates, addressing the common LLM issues of hallucination and lack of generalization [KVG+24; OSK+24b; SFHK25]. LLM2 synthesizes `domain.pddl` and `problem.pddl` files based on this rich, multi-source context. This modular pipeline (ASP $\rightarrow$ PDDL) is empirically shown to improve interpretability, error localization, and overall model robustness compared to direct LLM-to-PDDL generation [ASS24; OSK+24b].

For steps 4.4 to 4.7 refer Figure 4.3.

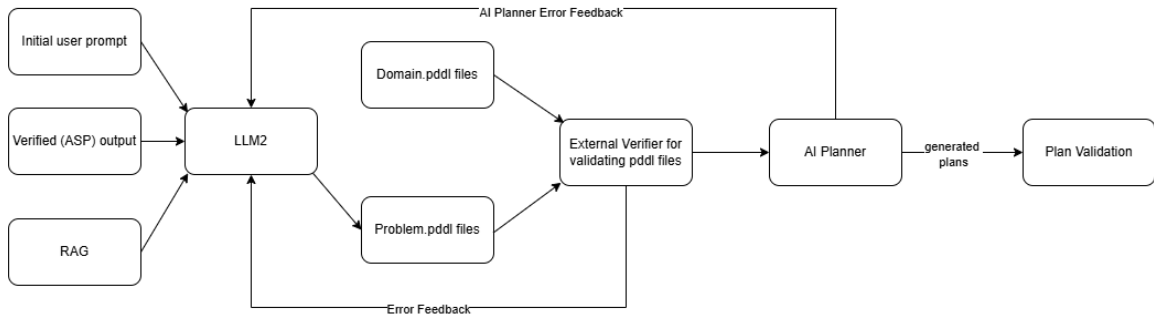


Figure 4.3: PDDL generation using RAG, ASP and generating plans

4.5 Syntactic Validation of PDDL and Iterative Correction

After generating the domain and problem files in PDDL format, we introduce a crucial validation step to make sure the models stick to the formal syntax of the planning language. This syntactic check acts as a protective measure, catching issues like undeclared predicates, undefined types, and poorly formed expressions.

To support this, the framework incorporates an external validation mechanism capable of parsing the generated models and producing structured diagnostic outputs. If the external verifier identifies any syntax or format errors, these diagnostics (including explicit error messages and faulty PDDL sections) are automatically formatted as prompts and sent back to LLM2 for correction using feedback loops. The LLM refines its output accordingly, and the loop continues until both `domain.pddl` and `problem.pddl` are validated as shown in Figure 4.3. This iterative refinement ensures that errors are eliminated early, preventing failures in planning or until we hit a set limit on the number of refinement attempts. By adding this layer of validation and correction early on, we reduce the chances of failures later in the process and boost the structural reliability of the models before they go into planning execution[ASS24; KVG+24].

4.6 Plan Generation and Semantic Feedback Loop

Following syntactic validation, the PDDL files are passed to a classical AI planner, which attempts to generate a valid plan for the given problem [Hel06].

- **Rationale:** Planning execution is the only way to guarantee that the generated PDDL models are not just syntactically correct but also operationally complete and goal-attainable—a limitation often observed in models generated by LLMs without plan-level feedback [OSK+24a; SJCG24a].
- **Semantic Feedback Loop:** As shown in Figure 4.3, if the planner fails to find a valid plan indicating possible semantic or modeling errors—its error messages, along with the PDDL files, are used to construct a targeted feedback prompt for LLM2. This feedback loop enables LLM2 to iteratively repair and refine the PDDL models until the planner successfully generates a plan.

4.7 Semantic Validation of the Generated Plans

While syntactic correctness of the domain and problem models is essential, it does not guarantee that the generated plan will be logically sound or executable. In order to determine whether the generated plan satisfies all preconditions, attains the desired goal states, and appropriately applies effects across state transitions, the last stage of the suggested pipeline incorporates an external plan validation step. This additional layer of verification ensures that even when the domain and problem files are syntactically valid, any latent semantic inconsistencies such as incorrect operator effects or missing causal links can be identified and addressed before model acceptance. Prior work such as [GVVK24] and [SFHK25] have highlighted the common failure of LLM-generated models in handling execution-level correctness, necessitating such semantic validation layers. [KVG+24]

further emphasized that LLMs, though expressive, lack grounding in formal planning semantics and must be paired with external verifiers for robust planning. The validator operates independently of the LLM, functioning as a neutral semantic checkpoint that enforces consistency between the intended logic and the realized behavior of the plan.

5 Implementation

The framework proposed for generating PDDL domain models from natural language descriptions is based on the cohesive coordination of several components during both the generation and verification phases. The implementation is described in detail in this chapter, including the prompts, choice and setup of the Large Language Models (LLM1 and LLM2), and the incorporation of external verification tools. The implementation is organized into sequential phases ranging from input processing, candidate construction, validation, refinement, to final synthesis to provide an overall reliable pipeline to facilitate syntactic correctness as well as semantic soundness of the produced models. The methodologies adopted at each phase are meant to reduce human interaction, facilitate reusability, and achieve the scalability of automated planning systems, finally enabling the production of reliable and planner-compliant domain models.

5.1 Prompt Design

Our approach begins with the use of domain-specific natural language prompts derived from AutoPlanBench¹, a publicly available benchmarking framework specifically designed to evaluate the planning capabilities of LLMs in classical planning domains. AutoPlanBench automatically generates domain encodings by extracting symbolic components such as actions, preconditions, and effects from PDDL domain files. These components are then mapped to templated natural language descriptions and refined using GPT-4, resulting in logically structured and fluent domain narratives.

A key strength of these prompts lies in their clarity and structure: they describe object types, available actions, preconditions for those actions, and their effects in a precise, step-by-step format. As shown by [SFHK25], these automatically generated prompts perform on par with the human-authored ones in terms of accuracy and consistency across planning tasks. Their stepwise nature facilitates model interpretability and supports structured reasoning, making them well-suited for use in neuro-symbolic planning pipelines.

AutoPlanBench contains a large collection of classical planning domains, many of which include multiple variants with different constraints or action formulations. For the purpose of this thesis, a representative but limited subset of base domains was selected at random. No domain variations were included in the evaluation to maintain consistency and avoid introducing confounding factors across experiments.

¹<https://github.com/minecraft-saar/autoplanbench>

In our framework, these prompts serve as a foundational input to LLM1, which uses them to generate ASP logic. The same prompt is reused in the second stage (LLM2) to support the generation of PDDL domain and problem files. This dual use maintains semantic consistency across intermediate representations, ensuring that the core meaning of the original task description is preserved throughout the pipeline.

5.2 CoT Strategy

To enable reliable logic generation from natural language, we integrate the domain prompts into a structured CoT prompting framework. This framework explicitly decomposes the reasoning task into sequential subtasks, guiding the LLM to process the domain description in a logically grounded manner. The CoT prompt used for LLM1 instructs the model to: (1) identify the initial and goal states, (2) extract all relevant object types and predicates, and (3) apply predefined transformation rules to convert natural language descriptions into ASP-compatible facts and rules.

These transformation rules are embedded directly into the prompt and are based on a set of hardcoded patterns implemented in our pipeline. They account for various linguistic and structural representations, such as atomic and relational predicates, grid-based spatial mappings, and time-indexed logic. In addition, the prompt enforces formatting constraints to ensure compatibility with Clingo, for instance, by specifying correct use of negation with ‘not’, grounding all variables, and ensuring consistent rule structure.

For LLM2, a similar CoT prompting strategy is employed to support PDDL domain and problem file generation. The prompt combines the validated ASP output, the original domain encoding from AutoPlanBench, and specific syntactic requirements for STRIPS-compliant PDDL. In cases where errors are detected by VAL or the AI planner, the pipeline reuses this prompt structure, augmented with the error message, to automatically refine the output. This feedback-driven refinement loop ensures that each generated model adheres not only to formal syntax but also to the intended semantics of the planning problem in order to generate valid plans.

By integrating structured CoT prompts at both stages, our approach achieves a high degree of reliability, enabling transparent reasoning, consistent transformation, and automated repair of logic representations.

As shown in Figure 5.1, the domain descriptions are given from the AutoPlanBench repository as the initial user input to the LLM.

Enter your planning domain description:

I can carry out the following actions:
 overcome a pain A with a pleasure B by using a food C from a planet D orbiting a planet E
 feast a pleasure A on a food B that eats a food C in a location E attacked by a location D
 succumb a pain A to a pleasure B that craves a food C in harmony with an object D orbiting an object E

I have the following restrictions on my actions:
 I can only overcome a pain A with a pleasure B by using a food C from a planet D orbiting a planet E if it is the case that A is a pain and B is a pleasure and A craves C and B craves C and C is food and B is in harmony with E and E is a planet and D orbits E and D is a planet
 I can only feast a pleasure A on a food B that eats a food C in a location E attacked by a location D if it is the case that A craves B and B is food and A is a pleasure and B eats C and C is food and B is located in E and D attacks E
 I can only succumb a pain A to a pleasure B that craves a food C in harmony with an object D orbiting an object E if it is the case that A fears B and A is a pain and B is a pleasure and B craves C and C is food and B is in harmony with D and D orbits E

The actions have the following effects on the state:
 Once I overcome a pain A with a pleasure B by using a food C from a planet D orbiting a planet E, it is the case that A fears B and B is in harmony with D
 Once I overcome a pain A with a pleasure B by using a food C from a planet D orbiting a planet E, it is not the case anymore that A craves C and it is not the case anymore that B is in harmony with E
 Once I feast a pleasure A on a food B that eats a food C in a location E attacked by a location D, it is the case that A craves C and B is located in D
 Once I feast a pleasure A on a food B that eats a food C in a location E attacked by a location D, it is not the case anymore that A craves B and it is not the case anymore that B is located in E
 Once I succumb a pain A to a pleasure B that craves a food C in harmony with an object D orbiting an object E, it is the case that A craves C and B is in harmony with E
 Once I succumb a pain A to a pleasure B that craves a food C in harmony with an object D orbiting an object E, it is not the case anymore that A fears B and it is not the case anymore that B is in harmony with D

Press Ctrl+Enter to apply

Generate Plan!

Figure 5.1: Domain description for Mystery98 Domain

5.3 Selecting LLMs and API keys

5.3.1 Choice of LLMs

Given the rapid growth of LLMs, it's important to establish a clear and systematic method for selecting which models to evaluate. For this thesis, we have chosen to focus on LLMs released after 2024, as they showcase the latest advancements in model structure, reasoning skills, and deployment efficiency [NKQ+23]. This choice keeps our evaluation relevant to current technologies and reflects the newest trends in LLM research and application.

5 Implementation

We started with a broad group of all known LLMs released in or after 2024. To narrow down this wide-ranging list and make our study practical and significant, we used a multi-step filtering process. The first filtering criterion focused on availability on the OpenRouter.ai platform, which is a popular open-source LLM gateway. It allows access to different models through unified API endpoints. This choice makes sense because OpenRouter.ai offers a consistent way to interact with models. This consistency is crucial for conducting comparative evaluations in research.

After filtering the list to include only those models available through OpenRouter.ai, we further reduced the selection by looking at the number of available variants for each LLM family available on the openrouter.ai for free and paid variants. The reason for this step is that LLM families with fewer variants are less fragmented. This makes it easier to compare their main capabilities without being influenced by internal scaling differences. By focusing on the three LLM families with the least variants on OpenRouter.ai, we created a more controlled and clear evaluation framework.

From this filtering process, we shortlisted the following three LLM families as seen in Figure 7:

- Gemini (developed by Google DeepMind)
- DeepSeek (developed by DeepSeek AI)
- Nemotron (developed by NVIDIA)

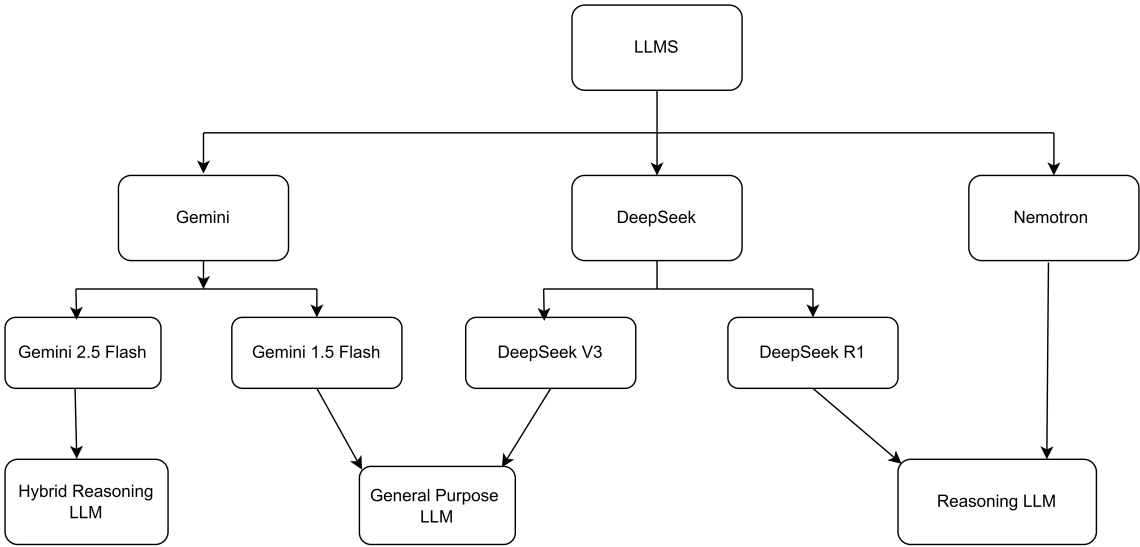


Figure 5.2: Selection of LLMS

However, we refined the selection to include only those variants that are freely accessible through API and match our research focus on model reasoning capabilities. Specifically, we chose only those variants that fit into one or more of the following categories:

- General-purpose models
- Reasoning-focused models
- Hybrid reasoning models

Based on these constraints, the final list of models considered in this thesis includes:

From the Gemini family:

- Gemini 1.5 Flash (General Purpose Model)
- Gemini 2.5 Flash (Hybrid Reasoning Model)

From the DeepSeek family:

- DeepSeek V3 (General Purpose Model)
- DeepSeek R1 (Reasoning Model)

From the Nemotron family:

- NVIDIA: Llama 3.3 Nemotron Super 49B v1 Super (Reasoning Model)

The 3 types of LLMs are:

- **Reasoning LLMs:** Reasoning LLMs are specialized models made for tasks involving multi-step logical reasoning, formal symbolic manipulation, program synthesis, and mathematical reasoning. These models are frequently evaluated against official benchmarks such as MATH or GSM8K and are trained on datasets that emphasize reasoning. For example, DeepSeek explicitly characterizes DeepSeek R1 as a logic-focused model that was trained with a robust focus on reasoning abilities, coding challenges, and sequential reasoning [DGY+25]. Similar to this, NVIDIA's Llama 3.3 Nemotron Super 49B v1 Super—particularly its Instruct variant—is regarded as a reasoning-optimized LLM designed to solve mathematical issues, generate code, and use tools. It performs exceptionally well on benchmarks that demand high-level reasoning [DGY+25]. Due to their structured outputs and reliability, these models are particularly suited for domains requiring formal correctness and logical rigor, such as neuro-symbolic systems.
- **Hybrid reasoning LLMs:** A hybrid reasoning model combines different reasoning methods to improve problem-solving performance. Traditionally, these models brought together rule-based reasoning (RBR), which uses clear logic rules, and case-based reasoning (CBR), which draws on past experiences or known cases to tackle new, similar problems. This combination helped expert systems work well even in areas with both clear and uncertain information. In modern neural language models (LLMs), hybrid reasoning means using designs or control methods that balance general language understanding with structured, logic-based reasoning like multi-step reasoning, code generation, or planning. These models can switch smoothly between natural language and symbolic manipulation, allowing them to handle both creative and formal reasoning tasks within one framework. This includes Gemini 2.5 Flash.
- **General purpose LLMs:** General-purpose LLMs are designed to perform a wide variety of tasks related to understanding and generating natural language. They work well across different areas. These models are trained on large and diverse datasets. Their main strength is their versatility and fluency. This allows them to handle applications like summarization, translation, dialogue generation, and creating content effectively. Two prominent examples in this category are DeepSeek V3 and Gemini 1.5 Flash.

DeepSeek V3 is a highly efficient, scalable universal LLM that uses a Mixture of Experts architecture. It selectively activates a small subset of its 236 billion parameters for each query. This setup allows for excellent language generation while being energy-efficient and adaptable for different tasks. DeepSeek V3 is a general purpose model optimized for both performance and flexibility in deployment [DLF+25].

Gemini 1.5 Flash, developed by Google, is described as a lightweight, latency-optimized version of the Gemini family. Even with its smaller computational footprint, it maintains strong contextual understanding and multimodal abilities. It is designed for fast, general-purpose inference across text, code, and image inputs. This makes it suitable for assistant-like tasks, real-time applications, and general NLP reasoning, though it is not specifically focused on complex symbolic tasks [TGL+24].

API Integration

The pipeline keeps track of each request's timestamp and applies wait periods as necessary to adhere to service-imposed rate constraints, which limit LLM to two requests per minute. By doing this, quota or throttling issues are prevented. Official libraries are used for all API interactions, and simultaneous processing of requests ensures dependability and facilitates debugging. The system has a solid error recovery process. If a request fails, it triggers a retry loop with exponential backoff, allowing the system to recover smoothly from temporary service interruptions or quota issues. Each LLM response is checked for correct structure and content before proceeding, which further increases reliability. This setup ensures secure, stable, and scalable LLM integration throughout all stages of the neuro-symbolic pipeline.

5.4 External Verifier and Plan Validator

In this implementation, the VAL (Validator) tool plays a dual but technically critical role, serving both as a syntactic checker for the generated PDDL models and as a semantic validator for the final plan. Initially developed by [HLF04] for the IPC, VAL has since become a standard for verifying both domain/problem files and the correctness of plans with respect to them. Its first role in this pipeline is to act as an external verifier immediately after LLM2 generates `domain.pddl` and `problem.pddl` files. VAL is used to perform strict syntactic validation, catching errors such as unmatched parentheses, missing colons, undefined predicates, or ill-formed parameters. The VAL error messages are parsed, formatted, and structured as targeted feedback prompts for LLM2 refinement if any such problems are found. This iterative correction loop continues until VAL confirms that both domain and problem models conform to expected PDDL syntax standards or has reached maximum iterations.

The second and equally important role of VAL occurs after the planning stage, where the AI planner has produced a plan based on the validated PDDL files. Here, VAL serves as a semantic validator, ensuring that every action in the plan respects precondition-effect constraints, achieves the stated goals, and adheres to causal links defined in the domain logic. This includes computing key plan metrics such as makespan and step-wise consistency, as demonstrated by works like [Pat24; SJCG24a], who used VAL not just for plan validation but also for computing execution feasibility,

cost, and timeline accuracy . Unlike the earlier syntactic verification stage, semantic plan validation does not initiate another LLM correction loop; instead, it serves as a final checkpoint. If a plan fails, it is not accepted as a valid outcome. This two-tiered integration mirrors best practices from recent literature, where symbolic tools like VAL serve as essential backstops for LLM-generated artifacts, bridging the gap between linguistic generation and logical execution. Overall, VAL provides a robust mechanism for elevating both the formal soundness and practical executability of the plans in this neuro-symbolic architecture.

5.5 AI Planner- Fast Downward

Fast Downward [Hel06] is a domain-independent planning system employed widely in automated planning research, particularly in benchmark environments like the IPC. It is a multi-stage planning architecture to enable search scalability in large planning spaces. Internally, it first transforms the PDDL problem to a finite-domain representation, then preprocesses it via causal graphs, domain transition graphs, and invariant synthesis, finally executing heuristic-guided search by means of selection methods such as Fast-Forward (FF) search or landmark heuristics.

Fast Downward has been widely used in recent studies that combine symbolic planning and LLMs. In the TIC pipeline by [al24a], Fast Downward is employed to validate PDDL domain models after logical compilation from intermediate ASP representations, ensuring that generated models are not only syntactically correct but also operationally sound. Similarly, [LHZ+23] show how useful Fast Downward is for connecting natural language inputs with executable plans by integrating it into their LLM+P methodology, which blends LLM-based translation with external planning validation. In neuro-symbolic pipelines with LLMs, Fast Downward performs a vital operational function in both plan generation and implicit validation. Recently, in works like those by [OSK+24a], [SJCG24a], as well as [VK23], Fast Downward was incorporated to check if LLM-generated domain and problem files can produce executable plans. When a planner fails to generate a plan even for syntactically correct PDDL, it's often a sign of semantic inconsistencies, i.e., unreachable goals, unmet preconditions, or conflicting effects, which syntax checkers like VAL might fail to identify.

In this thesis, Fast Downward was used at the backend stages of the pipeline. Once PDDL files have been checked by VAL for syntax, they are passed to Fast Downward. If no valid plan can be found, resulting error trace along with PDDL files as well as a fixing prompt are backprompted to LLM2 to suggest a revision to a domain model. The feedback loop allows for semantic refinement in an iterative manner, making the domain model not only valid in terms of its structure but also executable.

6 Experimental Evaluation

6.1 Benchmarks

The planning benchmarks used in this evaluation are sourced from the AutoPlanBench benchmark repository, which includes a wide range of classical IPC domains such as *Mystery98*, *Parking11*, *Zenotravel*, and others. These benchmarks were selected to cover diverse structural and semantic planning challenges.

A complete list of evaluated domains is provided in Table 6.2, where the performance of LLMs are analyzed across each domain.

6.2 Models

The five LLMs which are selected based on the selection criteria mentioned in the Implementation section are as follows:

- Gemini 1.5 Flash
- Gemini 2.5 Flash
- DeepSeek V3
- DeepSeek R1
- Nemotron Super

Every model was assessed based on its ability to automatically generate domain models and provide valid plans with minimal iteration.

6.3 Evaluation Metrics

Once the PDDL domain models have been refined and corrected through the validator-guided LLM feedback loop, they are then sent to a standard AI planner (Fast Downward) for generating plans.. The goal at this stage is to assess whether the models support the generation of valid, efficient, and executable plans, ensuring that the defined actions, predicates, and problem constraints can be effectively used by the planner to construct meaningful solutions that achieve the intended goals.

We use the following performance metrics, gathered for each domain and for each of the tested LLMs, to measure both correctness and optimization of the planning result:

6.3.1 Plan Generation Status

Once the PDDL domain and problem files have been refined via our validator-guided feedback loop, we submit them to Fast Downward and record whether it successfully produces a valid plan. A pass ✓ indicates that:

- Fast Downward parsed both files with no syntax errors or undefined symbols,
- the grounding phase completed without combinatorial explosion or semantic contradictions,
- and a sequence of actions reaching the goal state was found within our time and memory limits.
- VAL validated the generated plan.

A fail ✗ flags any issue in parsing, grounding, or search (e.g. “no plan found” or timeout), indicating further pipeline refinement is needed.

This binary metric gives a coarse measure of the pipeline’s end-to-end functionality, but a pass ✓ does *not* guarantee that the generated PDDL model is fully accurate with respect to ground-truth domains. For a deeper, logic-aware comparison against reference models, see Section 6.3.6.

6.3.2 Makespan (Sequential Plan)

Makespan refers to the total duration required to execute a sequence of actions in a plan. In the case of sequential plans, where actions are executed one after another with no overlap, the makespan is computed as the sum of the durations of all actions in the plan [HT23].

$$(6.1) \text{ Makespan} = \sum_{i=1}^n \text{duration}(a_i)$$

where:

- a_i is the i^{th} action in the plan,
- $\text{duration}(a_i)$ denotes the execution time of action a_i .

This metric is particularly useful for comparing refinements and selecting the most time-efficient domain models across different LLMs.

6.3.3 Plan Cost

Plan cost is the aggregated cost of all actions in a plan. This is computed using the VAL tool's metric report, extracted post-planning. Plan cost is an important metric for evaluating the efficiency of a plan in terms of resource utilization. This can be qualified in terms of action-specific resource usage, effort, or priority markings. Whereas makespan is time-oriented, cost is about resource effectiveness. Lower plan cost would normally predict fewer actions or more efficient use of planning resources [KSV+13].

Beneficial in analyzing areas with few resources available (e.g., fuel, payload, or energy restrictions).

Supplied explicitly by the VAL validator or derived from action metadata in the PDDL model.

6.3.4 Overall Time for Generating the Plan

This measure reflects the end-to-end time in seconds from the input of the initial LLM prompt to the generation of the planner-validated plan.

- LLM response time
- Number of refinement cycles
- Time for each validator (Clingo, VAL, Fast Downward)
- Planner execution time: It assesses the real-world usability and effectiveness of the complete pipeline.

6.3.5 Feedback Loop Count

This is a measure of how many iterations are required to improve a domain model until a workable plan is produced. Every loop could be started by:

- Syntactic errors, identified by either VAL (when validating only the domain and problem files) or by Fast Downward during its parsing stage. These include missing brackets, undeclared predicates, malformed `:goal` or `:init` blocks, and structural mismatches.
- Failures in plan generation occur when Fast Downward can parse the input but is unable to produce a valid plan. This includes scenarios like 'no solution found', 'goal not reachable', and 'initial state is a dead end'.

Monitoring this indicator gives us information on the quality of the LLM's original output as well as the efficacy of the refining logic. The LLM produced a nearly accurate model the first or second time since there were fewer feedback loops.

6.3.6 Accuracy

To determine how accurate the LLM-generated domain.pddl files are, we compare them to the groundtruth reference domains available in the AutoPlanBench benchmark repository. Previous research has introduced a variety of automated metrics for this evaluation. For example, the Action Reconstruction Error and Heuristic Domain Equivalence Metric developed by [OSK+24a] assess the exact symbolic and structural similarities between action schemas in both the ground truth and the generated domains. Similarly, [SJCG24a] and others check domain accuracy by ensuring that the entire planning and execution process works correctly with the generated model. However, these methods assume a strict symbolic alignment, meaning that predicate names, type names, and object constants must match exactly between the domain and problem files. Even small lexical differences—like `has_image` versus `have_image` or `pointing_at` versus `pointing`—can cause planner failures, making these evaluations overly sensitive and often unfair to LLM outputs that are logically correct but lexically different. Additionally, [GPQ+25] demonstrate that LLM-generated domains frequently struggle to generate valid plans, not because of flawed logic, but due to inconsistent naming between the domain.pddl and problem.pddl files, which standard planners see as undefined references.

To overcome this limitation, we’ve turned to a method that incorporates logic-aware predicate normalization along with fuzzy matching to evaluate accuracy. Our method is based on fuzzy lexical alignment, which uses the difflib library to calculate predicate name similarity ratios. This allows us to match tokens that differ in their phrasing but are conceptually comparable (`has_image` is basically equivalent to `have_image`, for example). In order to explicitly normalize known predicate variations as needed, we also keep an optional synonym map (for example, connecting `has_image` and `have_image` to a single term, `have_image`). In our evaluation, we extract all actions from both the groundtruth and generated domain files, calculating a weighted similarity score based on the similarity of action names and the Jaccard index for normalized preconditions and effects. This method allows for semantic alignment, even when there are variations in naming, resulting in a more lenient and realistic estimate of domain accuracy. Although this unique combination of fuzzy predicate matching and structural action comparison is new to our research, similar elements have been examined separately: [SJCG24a] suggest name normalization to enhance consistency in domain generation, and the TIC framework [al24a] aligns logic through intermediate representations to minimize symbolic mismatches. However, to the best of our knowledge, there hasn’t been a metric that directly compares generated and ground-truth domain.pddl files using both fuzzy matching and synonym normalization for evaluating structural similarity. Our method addresses this gap by delivering a scalable, interpretable, and semantically grounded accuracy score for LLM-generated PDDL domain models.

In Figure 6.1, you can see a practical example of how we compute and present this logic-aware accuracy evaluation. It details the results for the Satellite domain generated using the DeepSeek V3 model. The figure illustrates how the tool identifies and matches five actions between the LLM-generated domain.pddl and the corresponding ground-truth file. The average logic match score, which is derived from predicate-level fuzzy similarity across preconditions and effects, is an impressive 0.984, showing a nearly perfect alignment. We also provide individual similarity scores for each matched action, and because the average score exceeds the 0.85 threshold we established, we confidently return a final verdict of **MATCH**. This logic-aware evaluation gives us a more nuanced and semantically accurate measure of domain model accuracy than simply relying on strict symbolic comparisons.

```

1. Satellite Domain using DeepSeek V3
===== IMPROVED LOGIC-AWARE MATCH REPORT =====
Ground Truth Actions   : 5
Matched Actions        : 5
Average Logic Match    : 0.984
Individual Scores      : [0.92, 1.0, 1.0, 1.0, 1.0]
Verdict                : MATCH 
=====

```

Figure 6.1: Logic-aware match report for the Satellite domain using DeepSeek V3

6.4 Technical Configuration

To optimize and assess the effectiveness of the LLMs in automatically generating domain models, the technical configurations of the experimental setup employ a variety of methodologies for domain model generation, plan generation, and validation. The setup is hosted on some hardware and software configuration which are as follows:

Hardware Configuration

- **Operating System:** Windows 11
- **Processor:** 13th Gen Intel(R) Core(TM) i7-1360P @ 2.20 GHz
- **RAM:** 16 GB
- **GPU:** Intel(R) Iris(R) Xe Graphics

Software Configuration

- **VSCode:** Version (1.101.2), Visual Studio Code editor, employed for coding and integrating various tools and extensions.
- **Clingo:** Verifies syntactic correctness of ASP outputs
- **VAL:** Used for verifying syntax of PDDL files and validating plans and computing evaluation metrics like Makespan and Plan Cost.
- **Fast Downward planner:** Used for generating valid plans using domain.pddl and problem.pddl
- **Streamlit:** Hosts an interactive web dashboard for prompting the LLM, visualizing intermediate ASP and PDDL outputs, and monitoring validation feedback in real time.

6.5 Experimental Scenario

6.5.1 Observed Output Patterns and Error Taxonomy

Throughout the LLM-to-PDDL generation pipeline, we identified a range of output behaviors across the selected LLMs when given natural language descriptions of planning domains. These behaviors can be categorized into six distinct result types, each reflecting a unique aspect of model competence or failure. This classification allows for more nuanced evaluation beyond binary success metrics and highlights the importance of semantic alignment, plan feasibility, and symbolic validity. For complete implementation of the framework see Appendix A.

A. Fully Validated Plan Generation

This category captures successful end-to-end generation of valid domain and problem PDDL files along with a semantically validated plan.

A.1 Direct Success (First Attempt): The LLM generates correct and semantically valid PDDL files on the first attempt, requiring no additional refinement. These outputs are accepted by both the AI planner (e.g., Fast Downward) and the VAL validator. This indicates strong alignment between model priors and symbolic structure as seen in Figure 6.2.

Plan generated!

Final Plan Validation Output


```

Checking plan: D:\Srushti\Thesis\downward\benchmark11\sas_plan
Plan executed successfully - checking goal
Plan valid
Final value: 7

Successful plans:
Value: 7
D:\Srushti\Thesis\downward\benchmark11\sas_plan 7
    
```

Plan Validated with VAL!

Final Summary:

- Makespan: 7 steps
- Plan Cost: 7
- Total Generation Time: 144.10 seconds
- VAL Feedback Corrections: 0
- Planner Feedback Corrections: 0

Download All Files

Figure 6.2: Direct success without feedback loops

A.2 Success After Feedback-Based Refinement: The initial output is either syntactically incomplete or semantically incorrect, but after feeding validation errors back to the LLM from VAL as shown in Figure 6.3 and from Fast Downward as shown in Figure 6.4, the model successfully refines its outputs to generate valid plans. Figure 6.5 shows, feedback from both the external verifiers VAL and Fast downward. This illustrates the value of interactive feedback loops for neuro-symbolic repair. In many instances, the initial model output may contain partial or invalid constructs — e.g., missing predicates, ill-defined actions, or inconsistent goal declarations. However, when such errors are captured by external verifiers, and their feedback is used for backprompting the model, so the refined outputs become correct.

6 Experimental Evaluation

This feedback-driven correction loop is essential for:

- Fixing incomplete operator definitions
- Resolving action-effect contradictions
- Aligning variable typing and quantification

The screenshot displays a series of green status bars and a central validation output window. The first green bar indicates 'Planning succeeded after fixing domain/problem!'. The second green bar shows 'Plan generated!'. Below these is a light blue box titled 'Final Plan Validation Output' containing the following text: 'Checking plan: D:\Srushti\Thesis\downward\benchmark11\sas_plan', 'Plan executed successfully - checking goal', 'Plan valid', 'Final value: 13', 'Successful plans:', 'Value: 13', and 'D:\Srushti\Thesis\downward\benchmark11\sas_plan 13'. A third green bar states 'Plan Validated with VAL!'. At the bottom, a 'Final Summary:' section lists: 'Makespan: 13 steps', 'Plan Cost: 13', 'Total Generation Time: 208.09 seconds', 'VAL Feedback Corrections: 2', and 'Planner Feedback Corrections: 0'. A 'Download All Files' button is located at the very bottom.

✓ Planning succeeded after fixing domain/problem!

✓ Plan generated!

Final Plan Validation Output

Checking plan: D:\Srushti\Thesis\downward\benchmark11\sas_plan
Plan executed successfully - checking goal
Plan valid
Final value: 13

Successful plans:
Value: 13
D:\Srushti\Thesis\downward\benchmark11\sas_plan 13

✓ Plan Validated with VAL!

Final Summary:

- ✓ Makespan: 13 steps
- ✓ Plan Cost: 13
- ✓ Total Generation Time: 208.09 seconds
- ✓ VAL Feedback Corrections: 2
- ✓ Planner Feedback Corrections: 0

Download All Files

Figure 6.3: Success after feedback-based refinement with VAL

 Final Plan Validation Output

Checking plan: D:\Srushti\Thesis\downward\benchmark11\sas_plan
Plan executed successfully - checking goal
Plan valid
Final value: 4

Successful plans:
Value: 4
D:\Srushti\Thesis\downward\benchmark11\sas_plan 4

 Plan Validated with VAL!

 Final Summary:

-  Makespan: 4 steps
-  Plan Cost: 4
-  Total Generation Time: 225.41 seconds
-  VAL Feedback Corrections: 0
-  Planner Feedback Corrections: 1

 Download All Files

Figure 6.4: Success after feedback-based refinement with Fast-Downward

 Planning succeeded after fixing domain/problem!

 Plan generated!

 Final Plan Validation Output

Checking plan: D:\Srushti\Thesis\downward\benchmark11\sas_plan

Plan executed successfully - checking goal

Plan valid

Final value: 1

Successful plans:

Value: 1

D:\Srushti\Thesis\downward\benchmark11\sas_plan 1

 Plan Validated with VAL!

 **Final Summary:**

-  Makespan: 1 steps
-  Plan Cost: 1
-  Total Generation Time: 264.52 seconds
-  VAL Feedback Corrections: 2
-  Planner Feedback Corrections: 1

 Download All Files

Figure 6.5: Success after feedback-based refinement with VAL and Fast-Downward

A.3 Planner Fix Attempt Loop (Extended Case)

To further illustrate the effectiveness of planner-level backprompting, Figures 6.6 and 6.7 present a detailed example of a multi-step feedback loop triggered by an error detected during plan generation. In this case, the initial output from LLM2 included a predicate `in_harmony` that was used in the `problem.pddl`, but was never defined in the corresponding `domain.pddl`. This caused the Fast Downward planner to raise a parsing error, halting execution during the translation phase.

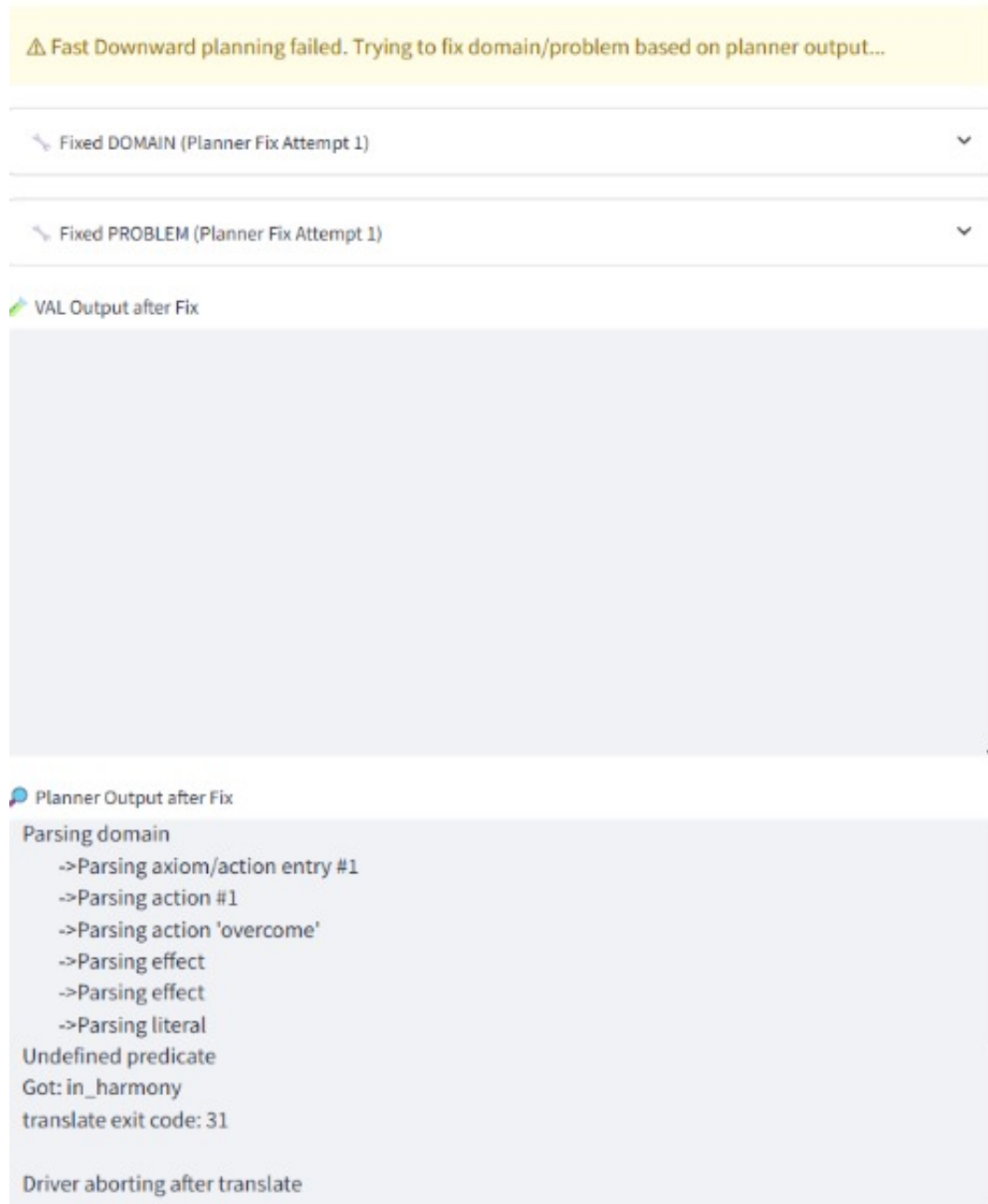


Figure 6.6: Fix Attempt 1: Planner fails due to undefined predicate “`in_harmony`”

6 Experimental Evaluation

The framework automatically captured this planner's output, extracted the relevant error message, and backprompted it into LLM2. The LLM2 was instructed to regenerate the `domain.pddl` and `problem.pddl` while addressing the undefined predicate. This error-specific correction process was completely automated and forms the core of our neuro-symbolic refinement mechanism. In the second iteration, the newly generated PDDL files correctly defined all necessary predicates and ensured internal consistency across domain and problem definitions.

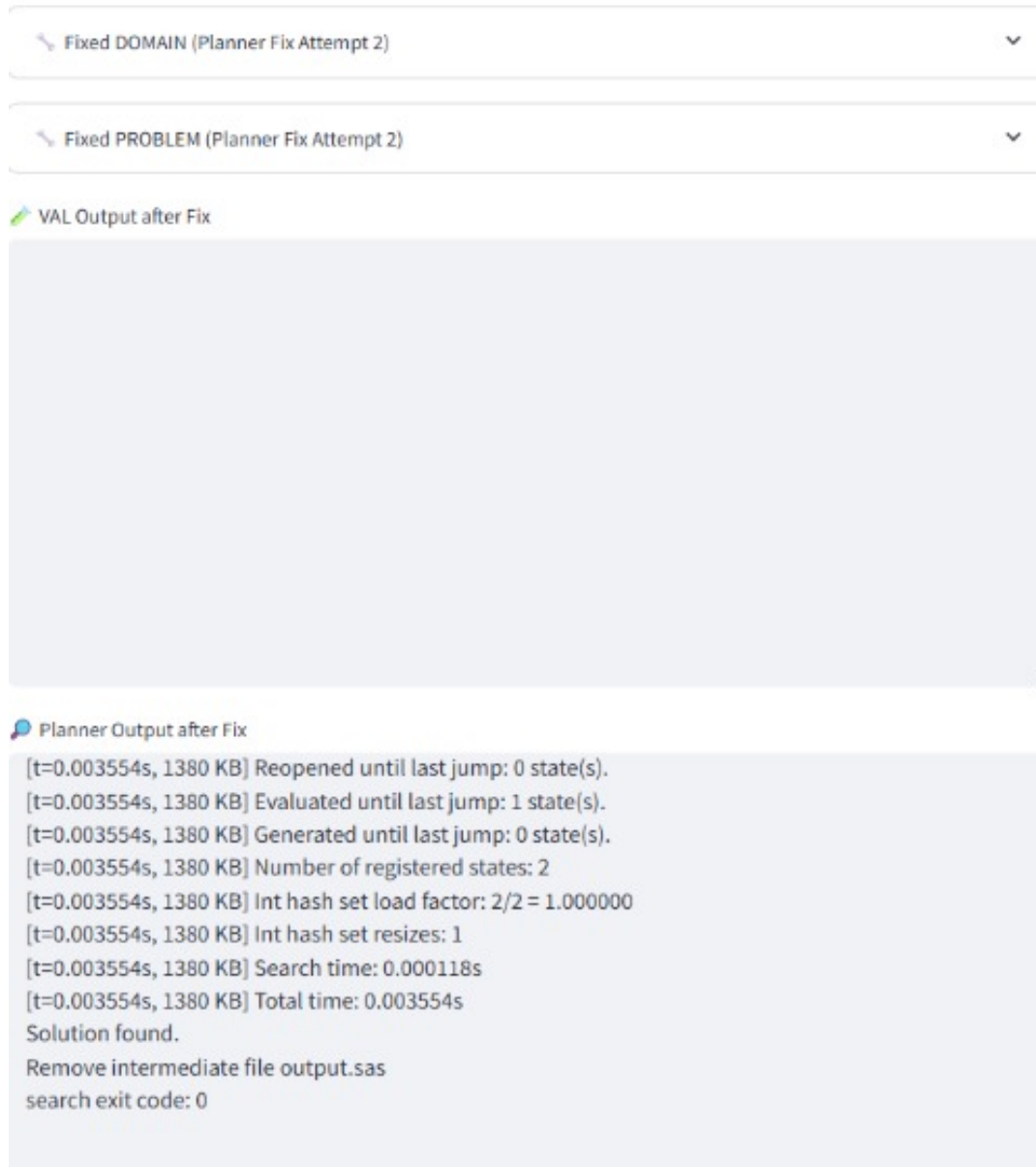


Figure 6.7: Fix Attempt 2: Planner successfully generates plan after feedback-based correction

As shown in Figure 6.7, Fast Downward was able to parse the regenerated files and successfully produce a valid plan. This example demonstrates not only the ability of the system to recover from symbol-level modeling errors but also highlights the importance of integrating planner diagnostics into the iterative generation loop. It shows how the planner functions not just as a consumer of

models, but also as an active participant in their validation and correction. Such multi-round refinement capabilities are critical for robust domain model generation in real-world settings where initial outputs are often incomplete or inconsistent.

B. Failure to Generate Valid PDDL

In this category, the LLM fails to produce syntactically or structurally valid PDDL files, even after multiple attempts as shown in Figure 6.8. These failures are detected during the initial validation phase by VAL, which is unable to parse the domain or problem file due to structural issues. Common failure modes include malformed syntax, missing parentheses, invalid predicate structures, or hallucinated natural language inside code blocks. This often reflects prompt misinterpretation.



Figure 6.8: Errors detected in PDDL files by VAL

C. VAL Verified PDDL files, but Planner Fails to Generate a Plan

As shown in Figure 6.9, in this case, the LLM produces syntactically and semantically valid PDDL files (confirmed by VAL), but the planner fails to find a solution. This may be due to unsolvable initial states, incorrect domain logic, or action schemas that prevent goal reachability. While less frequent, such cases point to deeper logical flaws in the modeled domain dynamics.

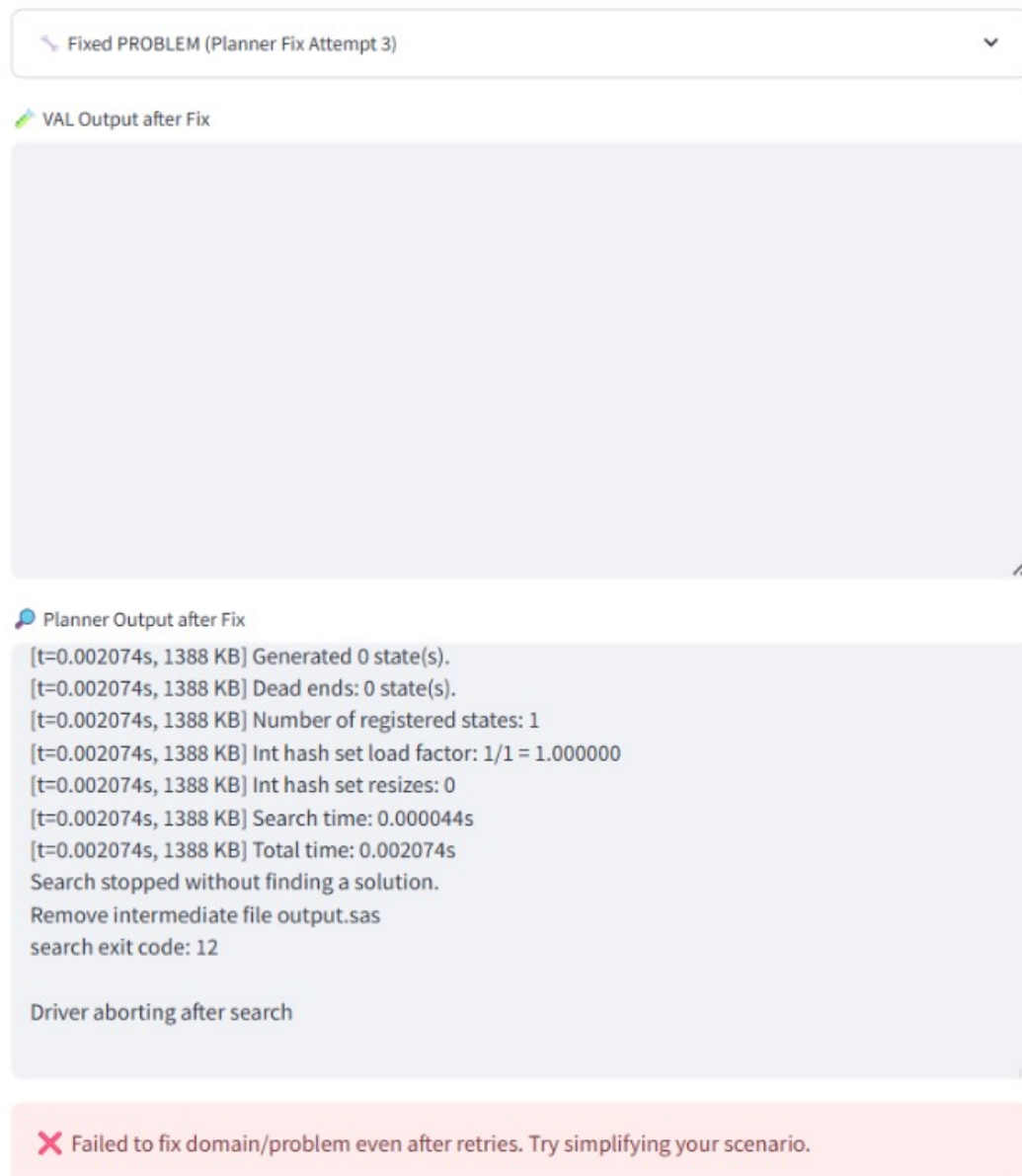


Figure 6.9: Errors detected in PDDL files during Plan generation

D. Plan Generation without Semantic Validation

Here, the LLM generates valid-looking PDDL files and the planner produces a plan. However, the plan fails semantic validation in VAL due to incorrect operator effects, unmet goal conditions, or invalid preconditions as shown in Figure 6.10. Although the outputs appear correct syntactically, they are logically inconsistent.

Examples of VAL failures include:

- Actions that violate preconditions

- Final goal conditions not being satisfied
- Unintended effects due to missing or overly generic constraints

 Planner Output

```
[t=0.002990s, 1388 KB] Reopened until last jump: 0 state(s).  
[t=0.002990s, 1388 KB] Evaluated until last jump: 16 state(s).  
[t=0.002990s, 1388 KB] Generated until last jump: 18 state(s).  
[t=0.002990s, 1388 KB] Number of registered states: 32  
[t=0.002990s, 1388 KB] Int hash set load factor: 32/32 = 1.000000  
[t=0.002990s, 1388 KB] Int hash set resizes: 5  
[t=0.002990s, 1388 KB] Search time: 0.000385s  
[t=0.002990s, 1388 KB] Total time: 0.002990s  
Solution found.  
Remove intermediate file output.sas  
search exit code: 0
```

 Plan generated!

 Final Plan Validation Output

 Plan Validation Failed!

Figure 6.10: Plan generated but failed validation

E. Valid but Semantically Irrelevant Output

A subtle but critical failure mode occurs when the LLM generates valid PDDL files and a valid plan, but the result is semantically irrelevant to the original domain description. This often happens after multiple feedback loops, where the model begins to overfit to validation errors and loses track of

the original intent. As a result, it may hallucinate unrelated actions or generate a plan for a different task altogether, passing all validations while being conceptually incorrect. Such cases can only be detected by comparing the generated domain and problem files against known ground truth PDDL definitions. This accuracy-based evaluation helps reveal discrepancies in goal structure, object usage, or action semantics that are not captured by VAL or the planner, but are essential for ensuring task fidelity.

F. Partially Valid Outputs with Hallucinated Elements

In some instances, the LLM mixes correct and incorrect information. For example, a valid domain definition may be paired with a hallucinated problem file containing non-existent objects or goal conditions. These partial successes signal instability in output consistency and often require multiple feedback iterations for resolution.

Summary of Result Types

Code	Result Type Description
A.1	Fully valid PDDL and plan on the first attempt
A.2	Fully valid outputs after iterative refinement
A.3	Planner Fix Attempt Loop (Extended Case)
B	No valid PDDL files generated
C	VAL verified PDDL files, but planner fails to generate a plan
D	PDDL and plan generated, but plan fails VAL validation
E	Valid plan and files, but irrelevant to original task/domain
F	Partial outputs with hallucinated or inconsistent elements

Table 6.1: Classification of LLM Output Behaviors

6.6 Analysis

For our thesis, we evaluate each LLM using a diverse set of 24 planning domains chosen from the AutoPlanBench GitHub repository. The selected domains vary in complexity and structural properties, ranging from simple STRIPS domains to those that include typed predicates, numeric fluents, and hierarchical dependencies.

As shown in the Table 6.2, We start with the Gemini family. Here 1st we have the analysis of general purpose LLM from the Gemini family, Gemini 1.5 Flash.

Gemini 1.5 Flash, our selected open-source general-purpose LLM from the Gemini family, demonstrated the most robust performance across the benchmark. As shown in the Table 6.2, this model successfully generated 18 out of 24 valid domain models. These domains not only passed syntactic validation but also produced semantically correct plans when tested with the Fast Downward planner and verified using VAL. This high success rate highlights the general robustness of the Gemini 1.5 Flash model in handling a diverse set of planning problems, even though it is not

Table 6.2: Evaluation of Domain Model Generation using Gemini 1.5 Flash

Domain	Plan Gen.	Makespan	Cost	Gen. Time (s)	VAL Fixes	Planner Fixes	Accuracy
barman14-bfs	✓	13	13	221.62	0	0	✗
blocksworld	✓	2	2	139.70	0	0	✗
childsnap14	✓	13	13	215.88	1	0	✗
depot	–	–	–	–	–	–	–
driverlog02-bfs	–	–	–	–	–	–	–
elevators08	–	–	–	–	–	–	–
ferry	✓	1	1	189.79	1	0	✓
floortile	✓	2	2	155.19	1	0	✓
goldminer	✓	3	3	127.03	0	0	✓
grid	✓	3	3	136.46	0	0	✓
gripper	✓	3	3	139.39	0	0	✗
hiking14	–	–	–	–	–	–	–
logistics	✓	26	26	126.03	0	0	✗
movie98	✓	7	7	144.10	0	0	✗
mprime98	–	–	–	–	–	–	–
mystery98	✓	1	1	264.52	2	1	✓
parking11	✓	4	4	225.41	0	1	✓
pegsol08	✓	2	2	137.44	0	0	✓
pipesworld-notankage04-bfs	✓	1	1	177.97	0	0	✗
rovers	–	–	–	–	–	–	–
satellite	✓	9	9	154.48	0	0	✓
scanalyzer08	✓	2	2	134.64	0	0	✓
visitall	✓	2	2	154.19	0	0	✓
zenotravel02	✓	3	3	152.14	0	0	✓

explicitly designed for symbolic reasoning. Its consistent structure, syntactic fluency, and balanced output generation capabilities make it a strong candidate for general-purpose symbolic translation tasks such as PDDL generation.

As illustrated in the Table 6.2, Gemini 1.5 Flash successfully generated valid plans for 18 domains. However, upon evaluating the correctness of the corresponding domain models against the ground truth, it was found that 7 of these 18 domains were inaccurate. Several factors may account for this discrepancy. In some cases, although the generated domain captured the majority of relevant elements, certain critical components were omitted. In other instances, the LLMs exhibited hallucination effects- producing domain models that were semantically unrelated to the given input description. A notable example of this behavior occurred with the *barman* domain: although syntactically correct PDDL files and valid plans were produced, the underlying domain model did not represent the *barman* domain. Instead, it more closely resembled that of the *logistics* or a structurally similar domain.

The second model, Gemini 2.5 Flash, is a hybrid reasoning LLM from the Gemini family, combining elements of structured reasoning with general NLP capabilities. According to the results in the Table 6.3, this model was able to successfully generated 5 out of 24 domain models that produced valid plans which were accepted by both the planner and the VAL validator. However, for the remaining 19 domains, its performance was mixed. While not entirely failing, the model still managed to generate plans for domains, but these were ultimately rejected by VAL due to semantic inconsistencies or incomplete action definitions. For the remaining domains, no valid plan could be

6 Experimental Evaluation

generated, indicating issues at the planning stage itself. These results suggest that while Gemini 2.5 Flash can demonstrate reasoning potential, its hybrid nature may sometimes lead to brittle outputs when high structural fidelity is required.

Table 6.3: Evaluation of Domain Model Generation using Gemini 2.5 Flash

Domain	Plan Gen.	Makespan	Cost	Gen. Time (s)	VAL Fixes	Planner Fixes	Accuracy
barman14-bfs	–	–	–	–	–	–	–
blocksworld	–	–	–	–	–	–	–
childsnaek14	–	–	–	–	–	–	–
depot	–	–	–	–	–	–	–
driverlog02-bfs	–	–	–	–	–	–	–
elevators08	–	–	–	–	–	–	–
ferry	–	–	–	–	–	–	–
floortile	–	–	–	–	–	–	–
goldminer	–	–	–	–	–	–	–
grid	✓	14	14	202.27	0	0	✓
grripper	–	–	–	–	–	–	–
hiking14	–	–	–	–	–	–	–
logistics	✓	3	3	288.72	0	0	✓
movie98	–	–	–	–	–	–	–
mprime98	✓	2	2	292.26	0	1	✓
mystery98	–	–	–	–	–	–	–
parking11	✓	3	3	181.48	0	1	✓
pegsol08	–	–	–	–	–	–	–
pipesworld-notankage04-bfs	✓	13	13	208.09	2	0	✗
rovers	–	–	–	–	–	–	–
satellite	–	–	–	–	–	–	–
scanalyzer08	–	–	–	–	–	–	–
visitall	–	–	–	–	–	–	–
zenotravel02	–	–	–	–	–	–	–

Upon conducting a thorough accuracy assessment of the 5 domains that initially produced valid plans, 4 were found to align correctly with the corresponding ground truth domain models. The remaining 1 domain, while syntactically valid and capable of producing executable plans, failed to capture essential aspects of the domain definitions, leading to semantic inconsistencies or incomplete representations.

Now moving toward the next LLM family, DeepSeek, we start with the general purpose model DeepSeek V3 as shown in the Table 6.4

DeepSeek V3, an open-source general-purpose model, showed moderate performance in our evaluations. As observed in the results Table 6.4, it successfully generated semantically valid plans for 14 out of 24 domain tasks. The model appears to be relatively stable in domains with straightforward logic and well-defined action sequences. However, for more complex domains requiring multi-step logical chaining or nested preconditions, its general-purpose architecture may have limited its ability to maintain coherent structures throughout the generated PDDL files.

Table 6.4: Evaluation of Domain Model Generation using Deepseek V3

Domain	Plan Gen.	Makespan	Cost	Gen. Time (s)	VAL Fixes	Planner Fixes	Accuracy
barman14-bfs	—	—	—	—	—	—	—
blocksworld	✓	2	2	175.58	0	0	✓
childsnaek14	—	—	—	—	—	—	—
depot	—	—	—	—	—	—	—
driverlog02-bfs	—	—	—	—	—	—	—
elevators08	—	—	—	—	—	—	—
ferry	✓	3	3	207.21	0	0	✓
floortile	—	—	—	—	—	—	—
goldminer	—	—	—	—	—	—	—
grid	✓	2	2	151.58	0	0	✓
gripper	✓	4	4	86.94	0	0	✓
hiking14	—	—	—	—	—	—	—
logistics	✓	3	3	145.81	0	0	✗
movie98	✓	2	2	60.21	0	0	✗
mprime98	✓	2	2	292.26	0	1	✓
mystery98	✓	2	2	325.58	0	1	✓
parking11	✓	0	0	210.25	0	0	✓
pegsol08	✓	1	1	100.81	0	0	✗
pipesworld-notankage04-bfs	—	—	—	—	—	—	—
rovers	—	—	—	—	—	—	—
satellite	✓	4	4	94.84	0	0	✓
scanalyzer08	✓	2	2	115.75	0	0	✓
visitall	✓	2	2	154.19	0	0	✓
zenotravel02	✓	3	3	154.14	0	0	✓

Despite this, DeepSeek V3 still demonstrated a reasonably strong capability in handling a subset of domain generation tasks, making it a viable option for general NLP-to-PDDL translation in less complex environments.

Out of the 14 domain models that yielded valid plans, only 11 matched the corresponding ground truth definitions upon close inspection. The remaining 3 failed the accuracy check.

The next model for analysis from the DeepSeek family is a reasoning model, DeepSeek R1.

DeepSeek R1 was selected as a reasoning-specialized model designed for logic-intensive tasks such as formal code generation and symbolic representation. Surprisingly, it was unable to validate even a single plan across the evaluated domains as shown in Table 6.5. While it was able to generate initial plans for domains, none of these passed the final semantic checks with VAL. The remaining domains failed even during the planning phase with Fast Downward. This is particularly notable given that DeepSeek R1 is promoted as a reasoning-capable model; however, our experiments reveal that it struggles with structural completeness and logical soundness required in PDDL. Even for simpler domains, iterative refinement using error feedback did not lead to valid plans. These findings suggest that reasoning models, while good at producing formal-looking output, may not always align well with strict symbolic execution environments unless explicitly fine-tuned for the planning domain.

It was also observed that DeepSeek R1, despite being designed as a reasoning-focused model, exhibited the longest generation time among all the evaluated LLMs. While Deepseek R1 being a reasoning model failed to produce any plans that could be validated using VAL, the model was still

Table 6.5: Evaluation of Domain Model Generation using Deepseek R1

Domain	Plan Gen.	Makespan	Cost	Gen. Time (s)	VAL Fixes	Planner Fixes	Accuracy
barman14-bfs	—	—	—	—	—	—	—
blocksworld	—	—	—	—	—	—	—
childsnaek14	—	—	—	—	—	—	—
depot	—	—	—	—	—	—	—
driverlog02-bfs	—	—	—	—	—	—	—
elevators08	—	—	—	—	—	—	—
ferry	—	—	—	—	—	—	—
floortile	—	—	—	—	—	—	—
goldminer	—	—	—	—	—	—	—
grid	—	—	—	—	—	—	—
gripper	—	—	—	—	—	—	—
hiking14	—	—	—	—	—	—	—
logistics	—	—	—	—	—	—	—
movie98	—	—	—	—	—	—	—
mprime98	—	—	—	—	—	—	—
mystery98	—	—	—	—	—	—	—
parking11	—	—	—	—	—	—	—
pegsol08	—	—	—	—	—	—	—
pipesworld-notankage04-bfs	—	—	—	—	—	—	—
rovers	—	—	—	—	—	—	—
satellite	—	—	—	—	—	—	—
scanalyzer08	—	—	—	—	—	—	—
visitall	—	—	—	—	—	—	—
zenotravel02	—	—	—	—	—	—	—

capable of generating both PDDL domain and problem files, and in several cases, it also succeeded in producing plans through the Fast Downward planner. However, these plans ultimately failed during the semantic validation phase, highlighting issues in logical consistency or incomplete domain definitions. The extended generation time may reflect the model’s effort to follow formal reasoning steps, yet this did not translate into higher-quality or verifiable outputs.

In contrast, DeepSeek V3, the general-purpose counterpart from the same family, demonstrated significantly faster generation times for both PDDL files and plans. Although it lacks explicit reasoning optimization, its outputs were often more concise and better aligned with the structural requirements of PDDL planning tasks. This suggests that a lightweight, general-purpose LLM like DeepSeek V3 may sometimes outperform heavier reasoning models in scenarios where speed and baseline correctness are prioritized over theoretical reasoning depth.

The 5th and the last LLM for the analysis was the Nemotron Super, again a reasoning model. The results are shown in the Table 6.6

Nemotron Super, another reasoning-specialized model developed by NVIDIA, was similarly evaluated across all benchmark domains. Nemotron Super failed to produce even a single valid plan throughout the evaluation similar to DeepSeek R1 which also is a reasoning model. The model struggled at multiple stages: in several domains, it was unable to generate a complete and syntactically correct plan, while in others, the plans it produced could not be executed due to semantic violations detected by the VAL validator. In fact, a significant number of tasks failed even earlier during the planning attempt, where Nemotron Super generated domain/problem files were rejected by Fast Downward due to structural inconsistencies or malformed definitions. However, our

Table 6.6: Evaluation of Domain Model Generation using Nemotron Super

Domain	Plan Gen.	Makespan	Cost	Gen. Time (s)	VAL Fixes	Planner Fixes	Accuracy
barman14-bfs	—	—	—	—	—	—	—
blocksworld	—	—	—	—	—	—	—
childsnaek14	—	—	—	—	—	—	—
depot	—	—	—	—	—	—	—
driverlog02-bfs	—	—	—	—	—	—	—
elevators08	—	—	—	—	—	—	—
ferry	—	—	—	—	—	—	—
floortile	—	—	—	—	—	—	—
goldminer	—	—	—	—	—	—	—
grid	—	—	—	—	—	—	—
gripper	—	—	—	—	—	—	—
hiking14	—	—	—	—	—	—	—
logistics	—	—	—	—	—	—	—
movie98	—	—	—	—	—	—	—
mprime98	—	—	—	—	—	—	—
mystery98	—	—	—	—	—	—	—
parking11	—	—	—	—	—	—	—
pegsol08	—	—	—	—	—	—	—
pipesworld-notankage04-bfs	—	—	—	—	—	—	—
rovers	—	—	—	—	—	—	—
satellite	—	—	—	—	—	—	—
scanalyzer08	—	—	—	—	—	—	—
visitall	—	—	—	—	—	—	—
zenotravel02	—	—	—	—	—	—	—

findings indicate that despite its reasoning capabilities, Nemotron Super lacks the domain-specific adaptability and semantic robustness required for classical planning tasks in PDDL. Even when iterative refinement was applied using detailed error feedback, the model was unable to converge on valid outputs.

7 Conclusion

This thesis presented a novel, validator-driven system that uses LLMs in a multi-stage neuro-symbolic pipeline to automatically generate domain models from natural language inputs. The architecture seeks to address key challenges faced by LLMs in producing valid PDDL models by incorporating RAG, intermediate ASP representations, and structured feedback loops through tools like Clingo, VAL, and Fast Downward. To evaluate the effectiveness of this feedback-driven system, five leading LLMs—Gemini 1.5 Flash, Gemini 2.5 Flash, DeepSeek V3, DeepSeek R1, and Nemotron Super—were tested for their capabilities in generating, refining, and validating domain models.

Rather than acting as mere debugging tools, the external validators served as learning aids, helping LLMs internalize structural and semantic rules over iterative refinements. Over time, this feedback loop yielded progressively more accurate and aligned outputs. In domains with complex dependencies, significant variability was observed across LLMs, as some models adapted quickly, while others required multiple iterations to converge. This demonstrates that the difficulty of the domain plays a crucial role in convergence behavior alongside model architecture. Validator feedback further improved interpretability by clarifying whether errors were structural (e.g., malformed action schemas) or conceptual (e.g., irrelevant goals).

The external verifier’s feedback mechanism functioned as both a correction tool and an iterative optimization layer. By narrowing the scope of necessary changes in each pass, the models incrementally improved toward valid outputs. These corrections often addressed upstream errors that might otherwise propagate into more severe inconsistencies. The feedback cycle acted as a fine-grained debugging process that evolved partial, broken representations into planner-compatible domain models.

Clingo, Fast Downward, and VAL together provided multi-layered validation. Clingo checked the logic structure of the ASP representation, while Fast Downward ensured that a valid plan could be produced from the domain/problem files. VAL then verified whether the plan truly satisfied the intended goals. While each tool served a distinct purpose, their combined use captured a broad range of logical and symbolic inconsistencies that a single validator alone might miss. A plan passing one stage could still fail in another, reinforcing the importance of layered validation.

Experimental results revealed strong variation in model performance based on architecture. Reasoning-optimized models like DeepSeek R1 needed fewer iterations to generate structurally correct outputs, but often failed semantic validation due to weak goal-effect alignment. In contrast, general-purpose models sometimes required more feedback loops but tended to produce plans that were both syntactically and semantically valid. Performance also varied by domain; some stabilized in one iteration, others in more. While no single model dominated across all metrics, general-purpose LLMs showed greater adaptability, while reasoning models exhibited structural precision with brittle semantics.

7 Conclusion

Despite these gains, the pipeline has limitations. Some errors persisted beyond the three allowed feedback iterations—particularly in complex domains—suggesting that a static iteration cap may prevent full convergence. Additionally, not all faults were evident in the initial passes; issues like missing type declarations or inconsistent preconditions sometimes appeared only in later outputs.

A promising direction for future work is the systematic evaluation of output consistency across repeated generations with identical prompts. Prior research has highlighted variability in LLM behavior, but there remains a lack of quantitative insight into how this affects planning reliability [al24b; KVG+24; OSK+24b]. This thesis observed that even when outputs initially passed validation, repeated runs with the same input could produce divergent models, differing in syntax, planner compatibility, and validation outcomes. In some instances, valid plans in one trial failed in the next, highlighting stochasticity in generation.

To mitigate this, future research should investigate large-scale empirical evaluations across domains and model types, tracking convergence patterns, error types, and divergence behavior. This includes identifying inconsistencies such as missing predicates, fluctuating action effects, and shifting type declarations. Potential solutions could include ensemble generation, structural confidence scoring, or constrained decoding with symbolic priors. These approaches could make LLM-based planning pipelines more robust and predictable, which is essential for deployment in high-stakes domains.

Another improvement would be to replace the fixed iteration limit with adaptive stopping criteria based on validator feedback or model uncertainty, allowing more flexibility and better convergence for complex or noisy domains. This could ensure that persistent structural or semantic issues are fully addressed, increasing the robustness of the generated domain models.

Bibliography

- [AF18] M. Asai, A. Fukunaga. “Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary”. In: *AAAI Conference on Artificial Intelligence*. 2018, pp. 6094–6101 (cit. on p. 33).
- [AFP+18] A. Arora, H. Fiorino, D. Pellier, M. Métivier, S. Pesty. “A Review of Learning Planning Action Models”. In: *The Knowledge Engineering Review* 33 (Nov. 2018). doi: [10.1017/S0269888918000188](https://doi.org/10.1017/S0269888918000188) (cit. on p. 25).
- [AG24] M. Aiello, I. Georgievski. *Introduction to AI Planning*. 2024. arXiv: [2412.11642](https://arxiv.org/abs/2412.11642) [cs.AI]. URL: <https://arxiv.org/abs/2412.11642> (cit. on pp. 21, 22).
- [AJO19] D. Aineto, S. Jiménez, E. Onaindia. *Learning STRIPS Action Models with Classical Planning*. 2019. arXiv: [1903.01153](https://arxiv.org/abs/1903.01153) [cs.AI]. URL: <https://arxiv.org/abs/1903.01153> (cit. on p. 25).
- [AJO22] D. Aineto, S. Jiménez, E. Onaindia. “A Comprehensive Framework for Learning Declarative Action Models”. In: *Journal of Artificial Intelligence Research* 74 (July 2022), pp. 1091–1123. doi: [10.1613/jair.1.13073](https://doi.org/10.1613/jair.1.13073) (cit. on p. 25).
- [al23] M. et al. “There and Back Again: Extracting Formal Models from Natural Language with LLMs”. In: *arXiv preprint arXiv:2311.01994* (2023) (cit. on p. 28).
- [al24a] A. et al. “Translate-Infer-Compile for Accurate Text-to-Plan Using LLMs and Logical Representations”. In: *arXiv preprint arXiv:2402.19469* (2024) (cit. on pp. 28, 45, 50).
- [al24b] S. et al. “Generating Consistent PDDL Domains”. In: *ICAPS* (2024) (cit. on pp. 27, 70).
- [ASS24] K. Agarwal, V. Sreepathy, S. Srivastava. “Translate-Infer-Compile for accurate text-to-plan using LLMs and Logical Representations”. In: *arXiv preprint arXiv:2403.02309* (2024) (cit. on pp. 16, 35, 36).
- [BET11] G. Brewka, T. Eiter, M. Truszczynski. “Answer set programming: An introduction to the special issue”. In: *AI Magazine* 32.3 (2011), pp. 78–80 (cit. on p. 23).
- [DGY+25] DeepSeek-AI et al. *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*. 2025. arXiv: [2501.12948](https://arxiv.org/abs/2501.12948) [cs.CL]. URL: <https://arxiv.org/abs/2501.12948> (cit. on p. 43).
- [DLF+25] DeepSeek-AI et al. *DeepSeek-V3 Technical Report*. 2025. arXiv: [2412.19437](https://arxiv.org/abs/2412.19437) [cs.CL]. URL: <https://arxiv.org/abs/2412.19437> (cit. on p. 44).
- [GA15] I. Georgievski, M. Aiello. “HTN planning: Overview, comparison, and beyond”. In: *Artificial Intelligence* 222 (Feb. 2015), pp. 124–156. doi: [10.1016/j.artint.2015.02.002](https://doi.org/10.1016/j.artint.2015.02.002) (cit. on p. 33).
- [Geo23] I. Georgievski. “Software Development Life Cycle for Engineering AI Planning Systems”. In: Jan. 2023, pp. 751–760. doi: [10.5220/0012149100003538](https://doi.org/10.5220/0012149100003538) (cit. on p. 15).

- [GKKS14] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub. *Clingo = ASP + Control: Preliminary Report*. 2014. arXiv: 1405.3694 [cs.PL]. URL: <https://arxiv.org/abs/1405.3694> (cit. on p. 23).
- [GKKS19] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub. “Multi-shot ASP solving with clingo”. In: *Theory and Practice of Logic Programming* 19.1 (2019), pp. 27–82 (cit. on p. 33).
- [GPQ+25] H. Guo, H. Pi, Y. Qin, Z. Tang, K. Li. *Leveraging Pre-trained Large Language Models with Refined Prompting for Online Task and Motion Planning*. 2025. arXiv: 2504.21596 [cs.R0]. URL: <https://arxiv.org/abs/2504.21596> (cit. on p. 50).
- [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: *arXiv preprint arXiv:2305.14909* (2023). URL: <https://arxiv.org/abs/2305.14909> (cit. on pp. 16, 17, 29, 35).
- [GVVK24] A. Gundawar, K. Valmeekam, M. Verma, S. Kambhampati. *Robust Planning with Compound LLM Architectures: An LLM-Modulo Approach*. 2024. arXiv: 2411.14484 [cs.CL]. URL: <https://arxiv.org/abs/2411.14484> (cit. on p. 36).
- [HDN12] A. Hertle, C. Dornhege, B. Nebel. “Plan repair in hybrid domains using semantic attachments”. In: *Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS)*. 2012 (cit. on p. 33).
- [Hel06] M. Helmert. “The Fast Downward Planning System”. Ph.D. thesis. Albert-Ludwigs-Universität Freiburg, 2006. URL: <https://www.fast-downward.org> (cit. on pp. 36, 45).
- [HH24] Y. Huang, J. Huang. *A Survey on Retrieval-Augmented Text Generation for Large Language Models*. 2024. arXiv: 2404.10981 [cs.IR]. URL: <https://arxiv.org/abs/2404.10981> (cit. on p. 22).
- [HLC+24] X. Huang, W. Liu, X. Chen, X. Wang, H. Wang, D. Lian, Y. Wang, R. Tang, E. Chen. “Understanding the Planning of LLM Agents: A Survey”. In: *arXiv preprint arXiv:2402.02716* (2024). URL: <https://arxiv.org/abs/2402.02716> (cit. on p. 27).
- [HLF04] R. Howey, D. Long, M. Fox. “VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL”. In: *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*. 2004, pp. 294–301. DOI: 10.1109/ICTAI.2004.120 (cit. on pp. 28, 44).
- [HLP+23] X. Huang, H. Lu, G. Pang, H. Tong, D. Tao. *Understanding the Planning of LLM Agents: A Survey*. 2023. arXiv: 2401.15513 [cs.AI]. URL: <https://arxiv.org/abs/2401.15513> (cit. on p. 16).
- [HT23] V.N. Hartmann, M. Toussaint. *Towards computing low-makespan solutions for multi-arm multi-task planning problems*. 2023. arXiv: 2305.17527 [cs.R0]. URL: <https://arxiv.org/abs/2305.17527> (cit. on p. 48).
- [JDF+12] S. Jiménez, T. De la Rosa, S. Fernandez, F. Fernández, D. Borrajo. “A review of machine learning for automated planning”. In: *The Knowledge Engineering Review* 27 (Dec. 2012). DOI: 10.1017/S026988891200001X (cit. on p. 26).

- [JYW+18] Y. Jiang, Q. Yang, Q. Wu, W. Chen, Y. Zhang. “Task-motion planning with semantic attachments: An extensible architecture for knowledge-based planning”. In: *IEEE Transactions on Automation Science and Engineering* 15.3 (2018), pp. 1032–1049 (cit. on p. 33).
- [KSV+13] M. R. Khouadjia, M. Schoenauer, V. Vidal, J. Dréo, P. Savéant. “Pareto-Based Multi-objective AI Planning”. In: *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*. 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 p. 49).
- [KVG+24] S. Kambhampati, K. Valmeekam, L. Guan, M. Verma, K. Stechly, S. Bhambri, L. Saldyt, A. Murthy. *LLMs Can’t Plan, But Can Help Planning in LLM-Modulo Frameworks*. 2024. arXiv: 2402.01817 [cs.AI]. URL: <https://arxiv.org/abs/2402.01817> (cit. on pp. 16–18, 26, 31, 33, 35, 36, 70).
- [LHZ+23] Y. Liu, Z. Huang, Q. Zhu, H. Sun, J. Feng. “LLM+P: From Natural Language to Plans with Large Language Models and PDDL”. In: *arXiv preprint arXiv:2309.07871* (2023). URL: <https://arxiv.org/abs/2309.07871> (cit. on p. 45).
- [Lif99] V. Lifschitz. “Answer Set Planning”. In: *Proceedings of the 16th International Conference on Logic Programming (ICLP)*. 1999, pp. 23–37 (cit. on pp. 23, 33).
- [LPP+20] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, D. Kiela. “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 9459–9474. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf (cit. on p. 22).
- [MGH+98] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, D. Wilkins. *PDDL—The Planning Domain Definition Language*. Tech. rep. CVC TR-98-003. Yale Center for Computational Vision and Control, 1998. URL: <https://www.aaai.org/Papers/AIPS/1998/AIPS98-077.pdf> (cit. on pp. 15, 22).
- [NGT04] D. Nau, M. Ghallab, P. Traverso. *Automated Planning: Theory & Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004. ISBN: 1558608567 (cit. on p. 15).
- [NKQ+23] H. Naveed, A. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Barnes, A. Mian. *A Comprehensive Overview of Large Language Models*. July 2023. DOI: [10.48550/arXiv.2307.06435](https://doi.org/10.48550/arXiv.2307.06435) (cit. on p. 41).
- [OSK+24a] J. Oswald, K. Srinivas, H. Kokel, J. Lee, M. Katz, S. Sohrabi. *Large Language Models as Planning Domain Generators*. 2024. arXiv: 2405.06650 [cs.CL]. URL: <https://arxiv.org/abs/2405.06650> (cit. on pp. 16, 22, 33, 35, 36, 45, 50).
- [OSK+24b] J. Oswald, K. Srinivas, H. Kokel, J. Lee, M. Katz, S. Sohrabi. *Large Language Models as Planning Domain Generators*. 2024. arXiv: 2405.06650 [cs.CL]. URL: <https://arxiv.org/abs/2405.06650> (cit. on pp. 26, 35, 70).

- [OWJ+22] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, R. Lowe. *Training language models to follow instructions with human feedback*. 2022. arXiv: 2203.02155 [cs.CL]. URL: <https://arxiv.org/abs/2203.02155> (cit. on p. 17).
- [Pat24] K. Patil. “LLMs for AI Planning: A Study on Error Detection and Correction in PDDL Domain Models”. Master Thesis, supervised by Dr. Ilche Georgievski. MA thesis. Universitätsstraße 38, D-70569 Stuttgart: Institute of Architecture of Application Systems, University of Stuttgart, Nov. 2024 (cit. on pp. 15, 16, 27, 44).
- [SATK22] T. Silver, K. Allen, J. B. Tenenbaum, L. P. Kaelbling. *PDDL Planning with Pretrained Large Language Models*. 2022. arXiv: 2210.03595 [cs.AI]. URL: <https://arxiv.org/abs/2210.03595> (cit. on pp. 21, 22, 33).
- [SFHK25] K. Stein, D. Fišer, J. Hoffmann, A. Koller. *Automating the Generation of Prompts for LLM-based Action Choice in PDDL Planning*. 2025. arXiv: 2311.09830 [cs.AI]. URL: <https://arxiv.org/abs/2311.09830> (cit. on pp. 32, 35, 36, 39).
- [SJCG24a] P. Smirnov, F. Joubin, A. Ceravola, M. Gienger. *Generating consistent PDDL domains with Large Language Models*. 2024. arXiv: 2404.07751 [cs.R0]. URL: <https://arxiv.org/abs/2404.07751> (cit. on pp. 15, 33, 36, 44, 45, 50).
- [SJCG24b] P. Smirnov, F. Joubin, A. Ceravola, M. Gienger. *Generating consistent PDDL domains with Large Language Models*. 2024. arXiv: 2404.07751 [cs.R0]. URL: <https://arxiv.org/abs/2404.07751> (cit. on p. 16).
- [TGL+24] G. Team et al. *Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context*. 2024. arXiv: 2403.05530 [cs.CL]. URL: <https://arxiv.org/abs/2403.05530> (cit. on p. 44).
- [TZM25] M. Tantakoun, X. Zhu, C. Muise. *LLMs as Planning Modelers: A Survey for Leveraging Large Language Models to Construct Automated Planning Models*. 2025. arXiv: 2503.18971 [cs.AI]. URL: <https://arxiv.org/abs/2503.18971> (cit. on p. 15).
- [VCM+15] M. Vallati, L. Chrapa, T. McCluskey, M. Roberts, S. Sanner, D. Kitchin. “On the effective configuration of planning domain models”. In: *Proceedings of the 29th AAAI Conference on Artificial Intelligence*. 2015 (cit. on p. 21).
- [VK23] K. Valmeekam, S. Kambhampati. “Large Language Models as Zero-Shot Planners: Revisiting Plans, Goals, and Hyperactions”. In: *arXiv preprint arXiv:2305.09626* (2023). URL: <https://arxiv.org/abs/2305.09626> (cit. on pp. 16, 17, 45).
- [VTS13] T. Vaquero, F. Tonidandel, J. R. Silva. “itSIMPLE: Towards an integrated design of planning domains”. In: *Knowledge Engineering Review* 28.2 (2013), pp. 215–230 (cit. on p. 33).
- [WTB+22] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, W. Fedus. *Emergent Abilities of Large Language Models*. 2022. arXiv: 2206.07682 [cs.CL]. URL: <https://arxiv.org/abs/2206.07682> (cit. on p. 23).
- [WWS+22] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. V. Le, D. Zhou. “Chain of thought prompting elicits reasoning in large language models”. In: *arXiv preprint arXiv:2201.11903* (2022) (cit. on p. 24).

- [ZZL+25] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.-Y. Nie, J.-R. Wen. *A Survey of Large Language Models*. 2025. arXiv: 2303.18223 [cs.CL]. URL: <https://arxiv.org/abs/2303.18223> (cit. on p. 23).

All links were last followed on July 15, 2025.

A Appendix

A.1 Implementation of Movies98 domain using Gemini 1.5 Flash

Enter your planning domain description:

I can carry out the following actions:
rewind the movie when the counter is at two hours
rewind the movie when the counter is at a time other than two hours
reset the counter to zero
get an object A which are chips
get an object A which is a dip
get an object A which is a pop
get an object A which is cheese
get an object A which are crackers

I have the following restrictions on my actions:
I can only rewind the movie when the counter is at two hours if it is the case that the counter is at two hours
I can only rewind the movie when the counter is at a time other than two hours if it is the case that the counter is at a time other than two hours
I can always reset the counter to zero.
I can only get an object A which are chips if it is the case that A are chips
I can only get an object A which is a dip if it is the case that A is a dip
I can only get an object A which is a pop if it is the case that A is a pop
I can only get an object A which is cheese if it is the case that A is cheese
I can only get an object A which are crackers if it is the case that A are crackers

The actions have the following effects on the state:
Once I rewind the movie when the counter is at two hours, it is the case that the movie is rewind
Once I rewind the movie when the counter is at a time other than two hours, it is the case that the movie is rewind
Once I rewind the movie when the counter is at a time other than two hours, it is not the case anymore that the counter is at zero
Once I reset the counter to zero, it is the case that the counter is at zero
Once I get an object A which are chips, it is the case that there are chips
Once I get an object A which is a dip, it is the case that there is dip
Once I get an object A which is a pop, it is the case that there is pop
Once I get an object A which is cheese, it is the case that there is cheese
Once I get an object A which are crackers, it is the case that there are crackers

Generate Plan!

Figure A.1: Movies98 input Natural Language domain description

Enter your planning domain description:

Once I get an object A which is a dip, it is the case that there is dip
Once I get an object A which is a pop, it is the case that there is pop
Once I get an object A which is cheese, it is the case that there is cheese
Once I get an object A which are crackers, it is the case that there are crackers

Generate Plan!

```
time(0..10).
object(chips).
object(dip).
object(pop).
object(cheese).
object(crackers).

init(counter_at_two_hours).
neg_counter_at_zero(0).
init(neg_movie_rewind).
init(neg_there_are_chips).
init(neg_there_is_dip).
init(neg_there_is_pop).
init(neg_there_is_cheese).
init(neg_there_are_crackers).

holds(P, 0) :- init(P).
holds(neg_P, 0) :- init(neg_P).

happens(rewind_movie_two_hours, T) :- time(T), holds(counter_at_two_hours, T).
happens(rewind_movie_other_time, T) :- time(T), not holds(counter_at_two_hours, T).
happens(reset_counter, T) :- time(T).
happens(get_chips, T) :- time(T), object(chips).
happens(get_dip, T) :- time(T), object(dip).
happens(get_pop, T) :- time(T), object(pop).
happens(get_cheese, T) :- time(T), object(cheese).
happens(get_crackers, T) :- time(T), object(crackers).
```

Figure A.2: ASP Generation part 1

```

happens(rewind_movie_two_hours, T) :- time(T), holds(counter_at_two_hours, T).
happens(rewind_movie_other_time, T) :- time(T), not holds(counter_at_two_hours, T).
happens(reset_counter, T) :- time(T).
happens(get_chips, T) :- time(T), object(chips).
happens(get_dip, T) :- time(T), object(dip).
happens(get_pop, T) :- time(T), object(pop).
happens(get_cheese, T) :- time(T), object(cheese).
happens(get_crackers, T) :- time(T), object(crackers).

holds(P, T+1) :- happens(A, T), effect(A, P), time(T), time(T+1).
holds(P, T+1) :- holds(P, T), not happens(A, T), effect(A, neg_P), time(T), time(T+1).
holds(neg_P, T+1) :- happens(A, T), effect(A, neg_P), time(T), time(T+1).
holds(neg_P, T+1) :- holds(neg_P, T), not happens(A, T), effect(A, P), time(T), ti

effect(rewind_movie_two_hours, movie_rewound).
effect(rewind_movie_other_time, movie_rewound).
effect(rewind_movie_other_time, neg_counter_at_zero).
effect(reset_counter, counter_at_zero).
effect(get_chips, there_are_chips).
effect(get_dip, there_is_dip).
effect(get_pop, there_is_pop).
effect(get_cheese, there_is_cheese).
effect(get_crackers, there_are_crackers).

goal(movie_rewound).
goal(there_are_chips).
goal(there_is_dip).
goal(there_is_pop).
goal(there_is_cheese).
goal(there_are_crackers).

#show happens/2.
#show holds/2.

```

✓ ASP Verified!

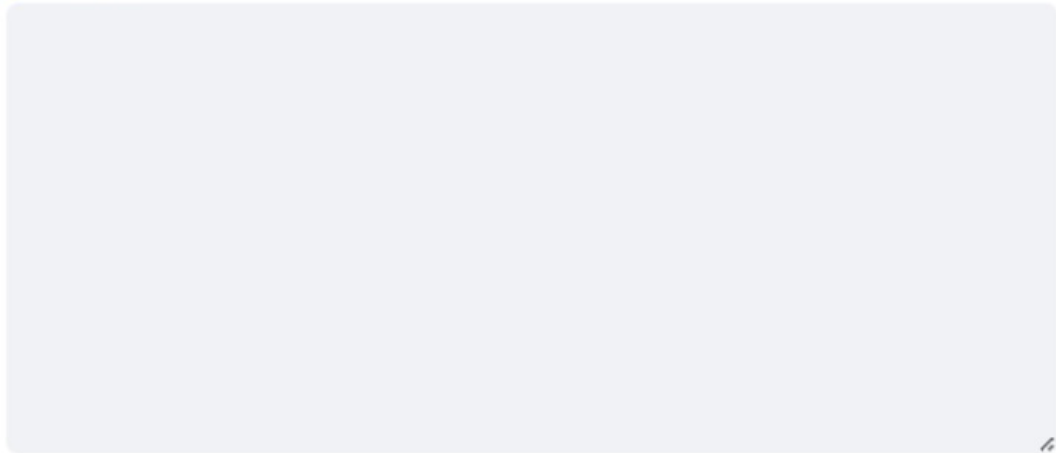
Figure A.3: ASP generation part 2

DOMAIN (Attempt 1)

PROBLEM (Attempt 1)

Figure A.4: PDDL generation started after ASP verified by Clingo

 VAL Output



 Planner Output

```
[t=0.002310s, 1300 KB] Reopened until last jump: 0 state(s).  
[t=0.002310s, 1300 KB] Evaluated until last jump: 1 state(s).  
[t=0.002310s, 1300 KB] Generated until last jump: 0 state(s).  
[t=0.002310s, 1300 KB] Number of registered states: 30  
[t=0.002310s, 1300 KB] Int hash set load factor: 30/32 = 0.937500  
[t=0.002310s, 1300 KB] Int hash set resizes: 5  
[t=0.002310s, 1300 KB] Search time: 0.000308s  
[t=0.002310s, 1300 KB] Total time: 0.002310s  
Solution found.  
Remove intermediate file output.sas  
search exit code: 0
```

 Plan generated!

Figure A.5: PDDL files verified by external verifiers and generated a plan

 Final Plan Validation Output

Checking plan: D:\Srushti\Thesis\downward\benchmark11\sas_plan
Plan executed successfully - checking goal
Plan valid
Final value: 7

Successful plans:
Value: 7
D:\Srushti\Thesis\downward\benchmark11\sas_plan 7

 Plan Validated with VAL!

 Final Summary:

-  Makespan: 7 steps
-  Plan Cost: 7
-  Total Generation Time: 144.10 seconds
-  VAL Feedback Corrections: 0
-  Planner Feedback Corrections: 0

 Download All Files

Figure A.6: Plan validated by 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