

**TRANSLATING HUMAN INSTRUCTIONS TO AI AGENTS USING
LANGUAGE MODELS**

by

XINYUE QIN (A0243468X)

A CAPSTONE PROJECT SUBMITTED FOR THE DEGREE OF

MASTER OF COMPUTING

in the

GENERAL TRACK

of the

NATIONAL UNIVERSITY OF SINGAPORE

2025

Supervisor and Examiner:
Professor Akshay Narayan

Declaration

I hereby declare that this project report is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in this report.

This report has also not been submitted for any degree in any university previously.

Xinyue Qin (A0243468X)

15 September 2025

To the Lovely World

Acknowledgments

I would like to express sincere thanks to my project supervisor – Professor Akshay Narayan, without his supportive and critical feedback during the four-month project period, I could not complete this and thus achieve a fulfilled result. His kind, patient and professional supervision broadens my vision and strengthen my capabilities in both academic and industrial fields, which without no doubt, would empower me with confidence and enthusiasm in my further career.

I would like to thank my dearest family and friends as well. The emotional support and always energetic encouragement kept me move forward whenever my spirits dipped. I wouldn't forget the days and nights you stand by with me. Thanks for being my strongest back up and I would always be the one that you can believe in and reply on.

Special thanks to the National University of Singapore (NUS) for providing essential resources and support during my postgraduate study period. Thanks to the professors of the master of computing program for their enriching teachings and support. The study method and thinking pattern I learned in NUS would always be a great treasure in my further life.

Thanks capstone coordinator Prasanna Karthik Vairam, capstone administrative staffs Ms. Geraldine Tang and Jin Xing, for helping me and other peers with the details regarding the project throughout the months. I am grateful to Professor Akshay Narayan for evaluating this project, and your feedback.

Hope all of us have the brightest further and live the life we want!

Contents

Acknowledgments	ii
Abstract	v
List of Figures	vii
List of Tables	viii
1 Introduction	1
1.1 Planning Domain Definition Language	1
1.2 Large Language Models	2
1.3 Fast Downwards	3
1.4 Overview	3
1.5 Problem Statement	3
1.6 Objectives	3
1.7 Summary of Contributions	3
2 Background and Literature Review	4
2.1 Related Work in Field1	5
2.2 Related Work in Field2	5
2.3 Existing Frameworks	5
2.4 Summary	5
3 Methods	6
3.1 PDDL Domains Experimented	6
3.2 NL Description Classes	7
3.3 Result Error Classes	8
3.4 Evaluation Methods Overview	9

4	Solution and Results	11
4.1	Base NL Description	11
4.1.1	Outputs and Evaluation of LLMs in Action Generation in Blocks World	11
4.2	High Level Overview	18
4.3	System Design	18
4.4	Implementation	18
4.5	Dataset	18
4.6	Evaluation	18
5	Conclusion and Future Work	19
5.1	Conclusion	19
5.2	Future Research Directions	19
6	Appendix	20
6.1	PDDL Domains	20
6.1.1	Blocks World Domain	20
6.1.2	Logistics Domain	21
6.1.3	Hiking Domain	22
6.1.4	Delivery Domain	24
6.2	PDDL Domains	25
	Bibliography	26

Abstract

Translating human instructions to AI agents using Language models

by

Xinyue Qin (A0243468X)

Master of Computing in

National University of Singapore

In recent advancements, large language models (LLMs) have exhibited proficiency in code generation and chain-of-thought reasoning. In our project, Planning Domain Definition Language (PDDL), as the standard encoding language for planning tasks, is used as the criterion to evaluate PDDL planning capabilities. We investigate whether large language models (LLMs) can be used to generate planning domain models from simple textual descriptions. Specifically, we introduce a framework for automated evaluation of LLM-generated domains by comparing the sets of plans for domain instances. Finally, we perform an empirical analysis of 7 large language models, including coding and chat models across 9 different planning domains, and under three classes of natural language domain descriptions. Our results indicate that LLMs, particularly those with high parameter counts, exhibit a moderate level of proficiency in generating correct planning domains from natural language descriptions. The working progress and project code are available at the project github page: https://github.com/Xinyue-qin/project_pri.

In recent advancements, large language models (LLMs) have exhibited proficiency in code generation and chain-of-thought reasoning, laying the groundwork for tackling automatic formal planning tasks. This study evaluates the potential of LLMs to understand and generate Planning Domain Definition Language (PDDL), an essential representation in artificial intelligence planning. We conduct an extensive analysis across 20 distinct models spanning 7 major LLM families, both commercial and open-source. Our comprehensive evaluation sheds light on the zero-shot LLM capabilities of parsing, generating, and reasoning with PDDL. Our findings indicate that while some models demonstrate notable effectiveness in handling PDDL, others pose limitations in more complex scenarios requiring nuanced planning knowledge.

These results highlight the promise and current limitations of LLMs in formal planning tasks, offering insights into their application and guiding future efforts in AI-driven planning paradigms.

List of Figures

List of Tables

4.1	Comparison between standard criteria (AIPS 2000) and Devstral generated actions by giving one-shot prompting and 'base' NL description of Blocksworld predicates.	13
4.2	Comparison between standard criteria (AIPS 2000) and Gemma3: 4b generated actions by giving one-shot prompting and 'base' NL description of Blocksworld predicates.	14
4.3	Comparison between standard criteria (AIPS 2000) and Gemma3: 12b generated actions by giving one-shot prompting and 'base' NL description of Blocksworld predicates.	15
4.4	Comparison between standard criteria (AIPS 2000) and Gemma3: 27b generated actions by giving one-shot prompting and 'base' NL description of Blocksworld predicates.	16
4.5	Distribution of four result error classes by giving one-shot, base description promptings in blocksworld. Lower is better for all classes except equivalent domain (Equiv), for which higher is better. No good results in bold.	17

Chapter 1

Introduction

1.1 Planning Domain Definition Language

Planning Domain Definition Language (PDDL) is the standard encoding language for "classical" planning tasks. The following are the components of a PDDL planning task, with definitions given respectively.

- Requirements: things in the world that interest us
- Objects: Entities in the world that can be acted upon
- Predicates: properties of objects that describe the world (can be true or false)
- Initial state: the state of the world that we start in
- Goal specification: things that we want to be true
- Actions/Operators: ways of changing the state of the world

For the convenience of illustration, the mathematical terminology that would be used is provided and symbolized as follows.

A lifted STRIPS planning problem is defined as a 5-tuple $\pi = \langle F, C, A, s_0, S_\star \rangle$. F is a finite set of predicates. C is a finite set of constants representing objects. We define F_g as the set of all grounded predicates, which indicates predicates in which all variables are replaced by legal constants from C . A state $s \subseteq F_g$ is a set of grounded predicates that describe the state, such that $f \in s$ if and only if f is a true fact about the world. S is the set of all possible states, which is the power set of F_g . A is a set

of action schema where each $a \in A$ is a 3-tuple $\langle pre(a), add(a), del(a) \rangle$, where $pre(a) \in F$ is the set of predicates that must hold to apply the action, $add(a) \subseteq F$ is the set of predicates that become true after the action is applied, and $del(a) \subseteq F$ is the set of predicates that become false. An action schema $a \in A$ can be grounded by substituting all variables in a with allowed constants from C . The grounded action $a_g = \langle pre_g(a_g), add_g(a_g), del_g(a_g) \rangle$ is defined as a 3-tuple of its grounded pre , add , and del predicates, and we define A_g as the set of all grounded actions. Finally, $s_0 \subseteq F_g$ is the initial state of the world for the planning task and $S_\star \subseteq S$ is the set of possible goal states.

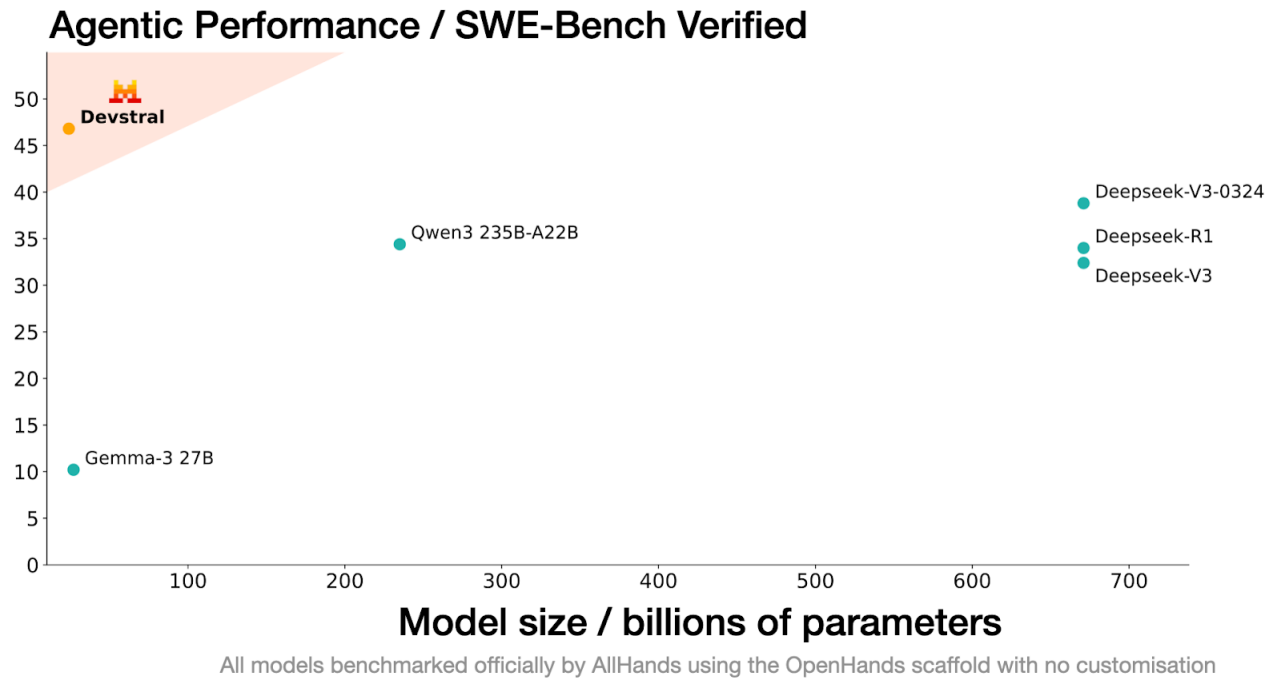
Planning tasks specified in PDDL are separated into three parts – domain, problem and plan. The **domain** contains predicates and actions that could be applied in a certain place. The **problem** includes objects, initial states and goal specification that could be performed in a defined space (i.e. domain). The **plan** that represents the effective set of actions to be run to achieve the goal.

Based on the aforementioned PDDL fundamental constitutions, we would investigate the LLMs capabilities in generating three above parts of a planning problem. i.e., action, problem and plan.

1.2 Large Language Models

Based on the consideration of LLMs We evaluate the Gemma3 family of LLMs ??, as well as Devstral ??.

Devstral is built under a collaboration between Mistral Small 3.1 and All Hands AI, therefore it has a long context window of up to 128k tokens. Devstral outperforms all open-source models on SWE-Bench Verified by a large margin, as illustrated by 1.2. It is the best open source model for coding agents with a compact size of just 24 billion parameters ??. For Gemma3 models, we evaluate the base models at 1b, 4b, 12b, 27b parameters.



1.3 Fast Downwards

1.4 Overview

1.5 Problem Statement

1.6 Objectives

1.7 Summary of Contributions

Chapter 2

Background and Literature Review

The paper written by Valmeekam et al. in 2022 ?? critiques the purported reasoning capabilities of large language models (LLMs) like GPT-3 and BLOOM, focusing on their ability to perform planning and reasoning about actions and change. The authors argue that existing benchmarks for evaluating LLMs' reasoning are overly simplistic and fail to assess more sophisticated cognitive tasks, such as goal-directed planning. They introduce an extensible assessment framework grounded in classical planning domains (e.g., Blocksworld) to evaluate LLMs across seven test cases, including plan generation, cost-optimal planning, and robustness to goal reformulation.

The results reveal that LLMs perform poorly on these tasks, with success rates as low as 0.6% for GPT-3 in plan generation, contrasting sharply with human performance (78% success in a preliminary study). The paper aligns with critiques by Marcus and Davis (2021) and others, questioning the depth of LLMs' reasoning. It contributes to ongoing debates about LLMs' limitations, emphasizing the need for more rigorous benchmarks to assess true reasoning capabilities beyond shallow pattern recognition.

This work underscores the gap between LLMs' language proficiency and genuine cognitive reasoning, calling for further research into hybrid systems combining LLMs with formal reasoning methods.

The paper "On the Planning Abilities of Large Language Models (A Critical Investigation with a Proposed Benchmark)" critically examines the planning capabilities of LLMs, addressing claims of their emergent reasoning abilities. The authors develop a benchmark based on International Planning Competition domains,

CHAPTER 2. BACKGROUND AND LITERATURE REVIEW

evaluating LLMs in three modes: autonomous, heuristic, and human-in-the-loop. Their findings reveal that LLMs perform poorly in autonomous plan generation (3% success rate), suggesting they lack robust reasoning about actions and change. However, they show promise in heuristic mode, where plans can be repaired by automated planners like LPG, and modestly improve human performance in collaborative settings.

The study contrasts with prior works like SayCan, which uses LLMs in heuristic mode for executable robotic plans, and highlights the limitations of zero-shot planning in commonsense domains. It also aligns with parallel research (e.g., Silver et al.) corroborating LLMs’ struggles with executable plans. The benchmark, made publicly available, aims to standardize evaluations of LLM planning abilities, addressing gaps in existing reasoning benchmarks (e.g., BIG-BENCH, GSM8K). The paper underscores the need for further research into whether LLMs learn causal models or rely on pattern matching, contributing to ongoing debates about their true reasoning capabilities.

[2] [3] [1]

2.1 Related Work in Field1

2.2 Related Work in Field2

2.3 Existing Frameworks

2.4 Summary

Chapter 3

Methods

3.1 PDDL Domains Experimented

In the project study, 4 PDDL domains with various difficulty and time scope are considered. Two classical domains from International Planning Competition 1998, "Blocksworld" and "Logistics", are taken into consideration. For recent domains, "Hiking" from PDDL 3.0 and "Delivery" (Yang et al. 2022) are the recent domains. It is worth mentioning that all of the four domains are built before Devstral and Gemma3 family models launched, which would standardize evaluation criteria, enhance reproducibility and eliminate the overfitting problem, and thus enable the evaluation of LLMs' underlying reasoning capabilities more fairly. These four domains are widely used in research studies and different planning scenarios. The detailed code for each domain could be found in the Appendix 6.1.

1. **Blocksworld (Difficulty level: easy)** - 5 predicates 4 actions: A robot hand tries to stack blocks on a table in a particular configuration.
2. **Logistics (Difficulty level: middle)** - 3 predicates 4 actions: Items must be transported to locations using planes and trucks.
3. **Hiking (Difficulty level: middle)** - 4 predicates 3 actions: Plan a hiking trip with dynamic constraints (weather, fatigue).
4. **Delivery (Difficulty level: hard)** - 7 predicates 3 actions: A delivery person must deliver newspapers to a number of safe locations from a home base.

3.2 NL Description Classes

In this project, to converting PDDL action $a \in A$ to the corresponding natural language (NL) descriptions $N(a) \in N(A)$, several strategies are investigated. Each strategy produces a distinct class of natural language representations of the action model. We would explore the influence of NL descriptions on LLMs action generation accuracy.

The following is an introduction of the three NL descriptions of interest in this project, with examples for better illustration.

1. Base $N_b(A)$: Base NL descriptions include only information including the action name, parameters, and parameter types of the action, as well as a one-line description of what the action does without explicitly mentioning any predicates. For example: The action "unstack" will have a hand unstack a block x from a block y .

2. Flipped $N_f(A)$: Flipped descriptions include the base descriptions with an additional description of all predicates that are deleted preconditions in that action schema, that is, for an action schema $a \in A$, $N_f(a)$ is $N_b(a)$ extended with a description of predicates in $pre(a) \cap del(a)$ as preconditions. The motivation behind this class is to evaluate if predicates that are explicitly changed are the most important things to include in a natural language description for the LLM, as they might be for a person when describing a domain. For example: 'The action 'unstack' will have a hand unstack a block x from a block y , if the block x is clear, on top of y , and the hand is empty.'

3. Random $N_r(A)$: Random descriptions act as a random baseline to compare against flipped descriptions, as well as another higher information content baseline to compare against base descriptions. For each action schema a , the description includes the base description $N_b(a)$, and descriptions of $|pre(a) \cup del(a)|$ random predicates sampled from $pre(a)$, $add(a)$ and $del(a)$, where the description is dependent on if the predicate was sampled from the precondition or effect. For example: 'The action 'unstack' will have a hand unstack a block x from a block y , if the hand is empty and x is on y . After the action, y should be clear.'

3.3 Result Error Classes

1. **Syntax Error:** The model produced syntactically invalid PDDL. This PDDL cannot be parsed to evaluate an action reconstruction error. Subclasses (in precedence order):

- (1) No PDDL (NoPDDL): Model did not output any PDDL;
- (2) Colon Error (ColonError): issues regarding the wrong use (missing, repetition) of colon in the PDDL;
- (3) Parenthesis Error (ParError): issues regarding the lack, or matching parenthesis in the PDDL;
- (4) Unexpected Token (UnToken): The PDDL parser failed after finding an unexpected token.

2. **Semantic Error:** The model produced syntactically valid PDDL, but the PDDL doesn't integrate with the intended problems. Subclasses:

- (1) Type Error (TErrors): The model produced an unexpected type;
- (2) Predicate Argument Error (PreError): the wrong number of variables were passed to a predicate;
- (3) Wrong Action Name (WroAcError): the name of the action is wrong;
- (4) Bad Precondition (BadPre): PDDL STRIPS does not allow negated preconditions, but one is present.

3. **Different Domain:** The model produced syntactically valid PDDL that integrates with the original domain, but the underlying domains are different by way of the domain equivalence heuristic. The behavior of the actions is not as intended, plans from the original domain cannot be applied in the new domain and vice versa. Subclasses:

- (1) No Plans Found (NoPlan): No plans were able to be found on problems in the new domain;
- (2) New Plan Application Error (NPError): Could not apply a new plan to the original domain;
- (3) Original Plan Application Error (BadPAError): The original plan could not be applied to the new domain.

4. **(Heuristically) Equivalent Domain (EquD):** The model produced syntactically valid PDDL that integrates with the desired domain under the domain

equivalence heuristic, plans from the original domain can be applied in the new domain and vice versa.

3.4 Evaluation Methods Overview

Based on the PDDL structures illustrated on chapter 1.1, we would investigate the capabilities of LLMs in generating action, problem and plan respectively. For context consistency, basic 'STRIPS' style actions are used in all requirements. Three NL description classes, i.e., base, flipped and random would be used in prompting input, for comparison.

The input and output for action generation task are shown in the signature 3.4.1. For illustration, the detailed prompts for LLMs to generate Blocks World action is given in the prompting 3.4. More detailed code, e.g., problem and plan generation code in several domains could be found in the appendix and the project github https://github.com/Xinyue-qin/project_pri.

```
- Task Signature = [input: base NL instruction, PDDL domain
predicates; output: PDDL action]
```

```
Given descriptions of each action in some domain, convert it to
Planning Domain Definition Language (PDDL) action. You may only
use the allowed predicates for each action.
```

```
Allowed Predicates:
```

```
- (arm-empty) : the robot arm is empty. - (holding ?x - block):
  block x is held.
- (clear ?x - block): block x is clear. - (on ?x - block ?y - block
  ) : block x is on block y.
- (on-table ?x - block): block x is on the table.
```

```
Output: The action, "putdown" will have the robot arm put down a
  block; the action, "pickup" will have the robot arm pick up a
  block; the action, "stack" will have the robot arm stack one
  block on another block; the action, "unstack" will have the
```

```
robot arm pick up a block that was stacked on another block;  
PDDL Action:
```

3.4.2 and 3.4.3 illustrate the problem and plan generation task signatures:

```
- Task Signature = Task Signature = [input: NL instruction, PDDL  
domain; output: PDDL problem]
```

```
- Task Signature = [input: PDDL domain, PDDL problem; output: Plan]
```

Chapter 4

Solution and Results

4.1 Base NL Description

4.1.1 Outputs and Evaluation of LLMs in Action Generation in Blocks World

4.1.1.1 Outputs of LLMs

The structure of input prompts and output is as followed:

```
- Task Signature = [input: base NL instruction, PDDL domain
  predicates; output: PDDL action]
```

Output of Gemma3: 1b is as follows:

(Syntax Error – No PDDL (NoPDDL): Gemma3: 1b model did not output any PDDL)

```
'''pddl
activate(robot_arm)
  where:
    robot_arm = {
      "arm-empty" = true
      "holding ?x - block" = true
      "clear ?x - block" = true
      "on ?x - block ?y - block" = true
```

```

        "on-table ?x - block" = true
    }
'''
**Explanation:**
*   'activate(robot_arm)': This indicates that the action is
    being performed by a robot arm.
*   'where:': This section defines the specific robot arm.
*   'robot_arm = { ... }': This defines the state of the robot arm,
    specifying the attributes:
    *   '"arm-empty" = true': The robot arm is empty.
    *   '"holding ?x - block" = true': The robot arm is holding block '?x'.
    *   '"clear ?x - block" = true': The robot arm is clear block '?x'.
    *   '"on ?x - block ?y - block" = true': The robot arm is on
        block '?x' at block '?y'.
    *   '"on-table ?x - block" = true': The robot arm is on the table
        at block '?x'.

```

This PDDL representation accurately captures the actions and their preconditions based on the provided descriptions.

4.1.1.2 Evaluation

We evaluate the capabilities of LLMs by replacing the actions in original domain with the LLMs' generated actions, and thus a new domain file is completed. By using Fast Downward to compile the modified domains and different variations and difficulty levels problems, we could figure out and classify the error into three result classes and 12 subclasses 3.3 by checking the log file.

By reviewing the performance of Devstral and Gemma3 family in the Blocks World PDDL domain file in generating PDDL actions with one-shot prompting, 'base' class NL instructions, a few results have been discovered.

Firstly, Gemma3: 1b makes poorest performance among all LLMs chosen. It makes the problem of 'unexpected token' since it could not generate code in PDDL syntax, not to mention adhere to the required action syntax.

In contrast, Devstral and most Gemma3 family (4b, 12b, 27b) could generate PDDL actions with complete and correct structures, which indicates key elements

CHAPTER 4. SOLUTION AND RESULTS

Devstral Generated Blocks World Actions	
pickup	putdown
<pre>(action pickup :parameters (?x - block) :precondition (and (on-table ?x) (clear ?x) (arm-empty)) :effect (and (not (on-table ?x)) (not (clear ?x)) (holding ?x)))</pre>	<pre>(action putdown :parameters (?x - block) :precondition (and (holding ?x)) :effect (and (not (holding ?x)) (on-table ?x) (clear ?x) (arm-empty)))</pre>
stack	unstack
<pre>(action stack :parameters (?x - block ?y - block) :precondition (and (holding ?x) (clear ?y)) :effect (and (not (holding ?x)) (on ?x ?y) (clear ?x) (arm-empty)))</pre>	<pre>(action unstack :parameters (?x - block ?y - block) :precondition (and (on ?x ?y) (clear ?x) (arm-empty)) :effect (and (not (on ?x ?y)) (not (clear ?x)) (holding ?x) (clear ?y)))</pre>

Table 4.1: Comparison between standard criteria (AIPS 2000) and Devstral generated actions by giving one-shot prompting and 'base' NL description of Blocksworld predicates.

CHAPTER 4. SOLUTION AND RESULTS

Gemma3: 4b Generated Blocks World Actions	
pickup	putdown
<pre> action pickup parameters: - x: block preconditions: - (clear ?x - block) effects: - (holding ?x - block) </pre>	<pre> action putdown parameters: - x: block preconditions: - (holding ?x - block) effects: - (arm-empty) - (clear ?x - block) </pre>
stack	unstack
<pre> action stack parameters: - x: block - y: block preconditions: - (holding ?x - block) - (on-table ?y - block) effects: - (on ?x - block ?y - block) - (arm-empty) </pre>	<pre> action unstack parameters: - x: block - y: block preconditions: - (on ?x - block ?y - block) effects: - (holding ?x - block) - (clear ?y - block) </pre>

Table 4.2: Comparison between standard criteria (AIPS 2000) and Gemma3: 4b generated actions by giving one-shot prompting and 'base' NL description of Blocksworld predicates.

CHAPTER 4. SOLUTION AND RESULTS

Gemma3: 12b Generated Blocks World Actions	
pickup	putdown
<pre>(define (action pickup :parameters (?x - block) :precondition (and (arm-empty) (clear ?x)) :effect (and (not (arm-empty)) (holding ?x) (not (clear ?x)))))</pre>	<pre>(define (action putdown :parameters (?x - block) :precondition (and (holding ?x)) :effect (and (not (holding ?x)) (clear ?x))))</pre>
stack	unstack
<pre>(define (action stack :parameters (?x - block ?y - block) :precondition (and (holding ?x) (clear ?y)) :effect (and (not (holding ?x)) (on ?x ?y) (clear ?x) (not (clear ?y)))))</pre>	<pre>(define (action unstack :parameters (?x - block ?y - block) :precondition (and (holding ?x) (on ?x ?y) (clear ?x)) :effect (and (not (holding ?x)) (clear ?y) (not (on ?x ?y)))))</pre>

Table 4.3: Comparison between standard criteria (AIPS 2000) and Gemma3: 12b generated actions by giving one-shot prompting and 'base' NL description of Blocksworld predicates.

CHAPTER 4. SOLUTION AND RESULTS

Gemma3: 27b Generated Blocks World Actions	
pickup	putdown
<pre>(define (action pickup :parameters (?x - block) :precondition (and (arm-empty) (on-table ?x) (clear ?x)) :effect (and (holding ?x) (not (on-table ?x)) (not (clear ?x))))</pre>	<pre>(define (action putdown :parameters (?x - block) :precondition (holding ?x) :effect (and (arm-empty) (on-table ?x)))</pre>
stack	unstack
<pre>(define (action stack :parameters (?x - block ?y - block) :precondition (and (holding ?x) (clear ?y)) :effect (and (on ?x ?y) (arm-empty) (not (holding ?x)) (not (clear ?y))))</pre>	<pre>(define (action unstack :parameters (?x - block ?y - block) :precondition (and (on ?x ?y) (clear ?x) (arm-empty)) :effect (and (holding ?x) (clear ?y) (not (on ?x ?y)) (not (arm-empty))))</pre>

Table 4.4: Comparison between standard criteria (AIPS 2000) and Gemma3: 27b generated actions by giving one-shot prompting and 'base' NL description of Blocksworld predicates.

CHAPTER 4. SOLUTION AND RESULTS

parameters, precondition and effect are included. the $pre(a)$, $add(a)$, and $del(a)$ these LLMs generate are overall logically correct. However, they could not produce completely parsable or equivalent actions because of failing to adhere to the required PDDL action syntax (ColonError). They lack colon before 'action'. For the partially parsable and solvable values, it is worthwhile noting that all these models have their contents very close, in other words the difference between them for a model is of few points. Under many-shot situations, these LLMs have high probability to generate parsable and solvable PDDL domains, which could be useful if the models are used as assistants for scientists.

Error Statistics in LLMs Action Generation					
Result Class	Devstral	Gemma3:1b	Gemma3:4b	Gemma3:12b	Gemma3:27b
NoPDDL	0	0	0	0	0
ColonError	5	0	5	5	5
ParError	0	0	0	0	0
UnToken	0	5	0	0	0
TError	0	0	0	0	0
PreError	0	0	0	0	0
WroAcError	0	0	0	0	0
BadPre	0	0	0	0	0
NoPlan	0	0	0	0	0
NPErrror	0	0	0	0	0
EquD	0	0	0	0	0

Table 4.5: Distribution of four result error classes by giving one-shot, base description promptings in blocksworld. Lower is better for all classes except equivalent domain (Equiv), for which higher is better. No good results in bold.

4.2 High Level Overview

4.3 System Design

4.4 Implementation

4.5 Dataset

4.6 Evaluation

Chapter 5

Conclusion and Future Work

5.1 Conclusion

5.2 Future Research Directions

Chapter 6

Appendix

6.1 PDDL Domains

6.1.1 Blocks World Domain

```
(define (domain blocksworld-4ops)
  (:requirements :strips)
  (:predicates (clear ?x - block)
               (on-table ?x - block)
               (arm-empty - block)
               (holding ?x - block)
               (on ?x ?y - block))

  (:action pickup
    :parameters (?ob)
    :precondition (and (clear ?ob) (on-table ?ob) (arm-empty))
    :effect (and (holding ?ob) (not (clear ?ob)) (not (on-table ?ob))
                 (not (arm-empty))))

  (:action putdown
    :parameters (?ob)
    :precondition (holding ?ob)
    :effect (and (clear ?ob) (arm-empty) (on-table ?ob) (not (holding
                                                                ?ob)))))
```

```

(:action stack
  :parameters (?ob ?underob)
  :precondition (and (clear ?underob) (holding ?ob))
  :effect (and (arm-empty) (clear ?ob) (on ?ob ?underob)
    (not (clear ?underob)) (not (holding ?ob))))

(:action unstack
  :parameters (?ob ?underob)
  :precondition (and (on ?ob ?underob) (clear ?ob) (arm-empty))
  :effect (and (holding ?ob) (clear ?underob)
    (not (on ?ob ?underob)) (not (clear ?ob)) (not (arm-empty))))

```

6.1.2 Logistics Domain

```

(define (domain logistics)
  (:requirements :strips :typing)
  (:types
    physobj
    object vehicle - physobj
    truck airplane - vehicle
    location - physobj
    airport - location
    city
  )
  (:predicates
    (loc ?l - location ?c - city)
    (at ?x - physobj ?l - location)
    (in ?p - object ?v - vehicle)
  )

  (:action load
    :parameters (?o - object ?v - vehicle ?l - location)

```

```

:precondition (and (at ?v ?l) (at ?o ?l))
:effect (and (in ?o ?v) (not (at ?o ?l)))
)

(:action unload
:parameters (?o - object ?v - vehicle ?l - location)
:precondition (and (in ?o ?v) (at ?v ?l))
:effect (and (at ?o ?l) (not (in ?o ?v)))
)

(:action drive
:parameters (?t - truck ?from - location ?to - location ?c -
city)
:precondition (and (at ?t ?from) (loc ?from ?c) (loc ?to ?c))
:effect (and (at ?t ?to) (not (at ?t ?from)))
)

(:action fly
:parameters (?p - airplane ?from - airport ?to - airport)
:precondition (at ?p ?from)
:effect (and (at ?p ?to) (not (at ?p ?from)))
)
)

```

6.1.3 Hiking Domain

```

(define (domain hiking)
  (:requirements :strips :typing)
  (:types
    hiker
    location
  )
)

```



```
(:predicates
  (at ?h - hiker ?l - location)
  (rained ?l - location)
  (has-energy ?h - hiker)
  (path-clear ?from ?to - location)
)

(:action walk
  :parameters (?h - hiker ?from ?to - location)
  :precondition (and
    (at ?h ?from)
    (path-clear ?from ?to)
    (not (rained ?to)))
  :effect (and
    (not (at ?h ?from))
    (at ?h ?to))
)

(:action rest
  :parameters (?h - hiker ?l - location)
  :effect (has-energy ?h)
)

(:action check-weather
  :parameters (?l - location)
  :effect (not (rained ?l))
)
)
```

6.1.4 Delivery Domain

```
(define (domain delivery)
  (:requirements :strips :typing)

  (:types
    agent package location - object
  )

  (:predicates
    (at ?a - agent ?l - location)
    (has ?a - agent ?p - package)
    (in ?p - package ?l - location)
    (connected ?from - location ?to - location)
    (requested ?p - package ?dest - location)
    (delivered ?p - package)
    (available ?a - agent)
  )

  (:action pick-up
    :parameters (?a - agent ?p - package ?l - location)
    :precondition (and (at ?a ?l) (in ?p ?l) (available ?a))
    :effect (and
      (has ?a ?p)
      (not (in ?p ?l))
    )
  )

  (:action move
    :parameters (?a - agent ?from - location ?to - location)
    :precondition (and (at ?a ?from) (connected ?from ?to) (
      available ?a))
    :effect (and
```

```
        (at ?a ?to)
        (not (at ?a ?from)))
    )

    (:action drop-off
      :parameters (?a - agent ?p - package ?l - location)
      :precondition (and (has ?a ?p) (at ?a ?l) (requested ?p ?l))
      :effect (and
        (delivered ?p)
        (in ?p ?l)
        (not (has ?a ?p)))
      )
    )
```

6.2 PDDL Domains

Bibliography

- [1] “Gemma 3 model overview | Google AI for Developers — ai.google.dev”, <https://ai.google.dev/gemma/docs/core>, [Accessed 28-07-2025].
- [2] K. Valmeekam, A. Olmo, S. Sreedharan, and S. Kambhampati, “Large language models still can’t plan (a benchmark for LLMs on planning and reasoning about change)”, in *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022. [Online]. Available: <https://openreview.net/forum?id=wUU-7XTL5X0>.
- [3] K. Valmeekam, S. Sreedharan, M. Marquez, A. Olmo, and S. Kambhampati, “On the planning abilities of large language models (a critical investigation with a proposed benchmark)”, 2023. arXiv: 2302.06706 [cs.AI]. [Online]. Available: <https://arxiv.org/abs/2302.06706>.