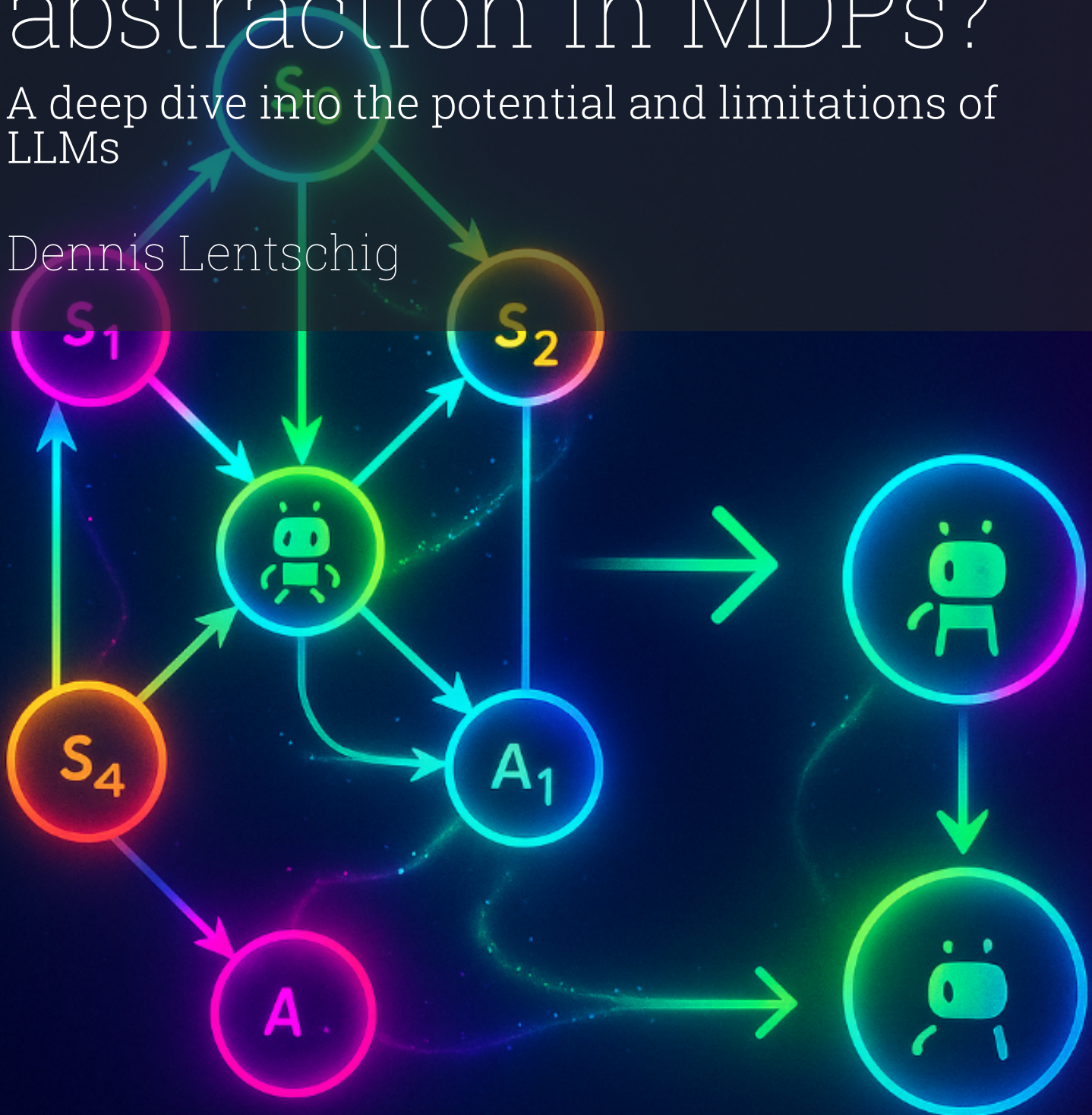
Can we use LLMs for abstraction in MDPs?

A deep dive into the potential and limitations of LLMs

Dennis Lentschig



Delft University of Technology

TU Delft

Can we use LLMs for abstraction in MDPs?

A deep dive into the potential and limitations of LLMs

by

Dennis Lentschig

Thesis Supervisor: Jinke He
Professor: Frans A. Oliehoek
Project Duration: June, 2024 - February, 2025
Faculty: Faculty of Electrical Engineering, Mathematics and Computer Science
Department: Intelligent Systems

Cover: MDP abstraction (generated by ChatGPT)
Style: TU Delft Report Style, with modifications by Daan Zwaneveld

TU Delft

Abstract

This thesis explores whether Large Language Models (LLMs) can generate abstractions in Markovian Decision Processes (MDPs) to reduce complexity in planning with Monte Carlo Tree Search (MCTS). A complete pipeline was developed to extract and validate cluster-based abstractions from LLMs. The pipeline combines modular prompt engineering, post-processing, and evaluation through both structural similarity and performance metrics. Experiments in gridworld environments show that Deepseek-R1 models consistently outperform LLaMA models, with architecture and training proving more important than parameter size. Structured prompts, especially those using JSON representation and rationale-driven responses, significantly improved abstraction quality. While LLMs can approximate, and sometimes even find the ideal abstractions in simple environments, performance deteriorates in larger or less regular domains. These findings highlight both the potential and current limitations of LLM-based abstraction, and suggest directions for future research, including more complex environments, richer abstraction types, and advanced prompting strategies.

The entire source code for the thesis can be found here

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
4X	eXploration, eXpansion, eXploitation and eXtermination
ARC	Abstract and Reason Corpus
DRL	Deep Reinforcement Learning
EMCTS	Elastic Monte Carlo Tree Search
EMD	Earth Mover's Distance
LLaMa (llama)	Large Language Model Meta AI
LLM	Large Language Model
LLMGA	LLM-based Game Agent
MAE	Mean Absolute Error
MCTS	Monte Carlo Tree Search
MCTSr	Monte Carlo Tree Self-Refine
MDP	Markovian Decision Process
MSE	Mean Squared Error
POMDP	Partially Observable Markovian Decision Process
RL	Reinforcement Learning
TBS	Turn-based Strategy Game
DHPC	Delft High Performance Computing Center

Symbols

Symbol	Definition
A	Finite set of actions
P	Transition function
$P^a_{ss'}$	Next state distribution -
R	Reward function
R^a_s	Expected immediate reward
S	Finite set of states
a	Action
s	State
γ	Discount factor
π	Policy
π^*	Optimal policy

1

Introduction

"The good thing about computers is that they do what you tell them to do. The bad news is that they do what you tell them to do." is a quote read to the player in Civilization 6 upon researching computers. Civilization 6 is a 4X game, a sub-genre of turn-based strategy games (TBS), which is an abbreviation for eXploration, eXpansion, eXploitation and eXtermination [1]. While Chess already has a very large state space, with upper bound estimates being 13^{64} [2], 4X games contain a considerable number of actions within a turn, the order of actions is important as previous actions can influence latter ones and the branching factor can significantly increase up to billions next states. Compared to games like Go and Chess, the branching factor is much higher in turn-based strategy games, and due to their length and interwoven-ness, create significantly more complex environments [3]. As such, these games can pose a significant challenge for AI agents as there is a lot of information to process via observations and previous actions.

In order to deal with highly complex environments, such as 4X games or even board games, it is necessary to help the agent during their decision-making process by simplifying it. The simplification comes in form of a mechanism called abstraction. While there is a plethora of different methods for abstraction [4] [5], a key similarity shared between all of them is helping the decision maker distinguish between relevant and irrelevant information. This in turn, helps reduce the complexity, as the agent only focuses on the relevant information to solve the environment. At its core, abstraction provides the means to an agent to reduce the complexity, however it is important to also maintain good decision quality. Applications like these are also not solely restricted to games, but can be applied to other areas as well, such as robotics or autonomous driving, emphasizing their importance.

These decisions and complex environments can be modeled using a Markovian Decision Process (MDP), as they reflect sequential decision-making when outcomes are uncertain. While typically rule-based or neural-network agents have been used for applications of determining abstractions, a new player has stepped on the field. In recent years, Large Language Models (LLMs) have revolutionized numerous fields, and have shown to have incredible capabilities when it comes to understanding and generating natural language. One of the main advantages of these LLMs against traditional rule-based or neural-network agents is the ability to engage in interactive conversation and textual reasoning [6]. Additionally, they have the capability to adaptively react and perform tasks based on the environment without predefined explicit instructions due to them being trained on larger datasets, enabling them to use their world knowledge [6]. However, they also suffer greatly from hallucinations [7] [8] and have shown to have difficulties with planning [9] [10] and object cohesion [11].

The aim of this thesis is to help bridge the gap between LLMs and abstraction in MDP environments. As found during a literature review, there had been plenty of uses of LLMs for games, also known as LLM-based game agents (LLMGAs) [7] [12] [13] [14] [15], however there are only few examples in which they are used in conjunction with Monte Carlo Tree Search (MCTS) [10] [8] and none, from personal findings, that involve the LLM dictating the abstractions. As such, this thesis aims to simplify the decision-making process for MCTS by abstracting the state space in a given environment using

LLMs.

This leads to the main research question that this thesis aims to answer:

Can we use LLMs for abstraction in MDPs?

This research question is however very general and not directly researchable. Therefore, two sub-questions were determined, in order to help quantitatively answer it. In short, this means they were used to determine metrics to evaluate the abstractions generated by the LLM.

- Can LLMs produce abstractions that are close to the optimal abstraction, if it exists?
- Can LLMs produce useful abstractions for planning?

Leading on, in order to adequately answer these questions and to systematically design the experiments, relevant subresearch questions (SRQs) were determined. The SRQs focus on specific areas of the experiments, such as LLMs, prompts and maps.

- **LLMs**
  - How does LLM type and size affect abstraction quality?
- **Prompts**
  - How does phrasing (prompt composition) affect the quality?
  - Is there a way to represent such an environment in the most optimal way for LLMs?
- **Maps**
  - How does the abstraction quality change with increasing task difficulty?
  - How does the abstraction quality change if there is no exact abstraction?

In this work, a complete pipeline was developed that uses LLMs to extract cluster-based abstractions for simplified MDP tasks, and validate these abstractions through both structural and planning performance metrics. Although the problem is framed in terms of general MDPs, a focus is put on the form of grid-based spatial environments, which serve as structured, interpretable benchmarks to evaluate abstraction quality. Thesis presents the following contributions:

- **Framework for abstraction extraction** - the thesis presents a simple framework that enables the extract of usable cluster-based abstractions from LLMs for MDPs.
- **Evaluation of abstractions** - the thesis provides a way to evaluate and benchmark generated abstractions based on a model-based and performance-based metric.

In order to present the findings, summarize the approaches and discuss the findings, the thesis is structured as follows. Firstly, chapter 2 provides context to relevant concepts mentioned in this thesis, whilst chapter 3 summarizes of the related work regarding LLMs and decision-making. Leading on, chapter 4 highlights how abstractions are extracted from LLMs and how abstractions are evaluated. Following this, chapter 5 contains the experimental design, experiments and their results. Lastly, a conclusion is provided in chapter 6. An appendix is provided for further source code.

2

Background

This chapter introduces the theoretical foundations and technical components relevant to this thesis. It provides a formal overview of MDPs, abstraction methods, similarity metrics, MCTS, and LLMs setting the stage for understanding the experiments and their interpretation. It is assumed that the reader has a basic understanding about the inner workings of LLMs prior to this.

2.1. Markovian Decision Processes

A MDP is a mathematical tool used to model decision making processes. An MDP M can be described using a tuple of 5 variables $M = \langle S, A, P, R, \gamma \rangle$ where

- S is a finite set of states
- A is a finite set of actions
- P is the transition function, where $P^a_{ss'}$ is the next-state distribution after doing action a in state s
- R is the bounded reward function, where R^a_s is the expected immediate reward gained from doing action a in state s
- γ is the discount factor

[16] [4] [17] [5].

Solutions can be found for an MDP which is called a policy and is denoted by $\pi(s)$. The policy tells the agent which action a to choose given a state s. The state-value function $V(s)$ contains the sum of all rewards that were earned. Using this, the most interesting solution can be found, namely the optimal policy π^*. The optimal policy is one that maximizes the state-value function $V^\pi(s)$, meaning $V^* = \texttt{max}_\pi V^\pi$ [4].

At this point it is important to note that the research done in **this thesis solely focuses on MDP environments**. Partially observable environments, in which an agent cannot observe the state it is in and therefore has to rely on observations, is not considered. This was due to the fact that Partially Observable Markovian Decision Process (POMDP) environments are already significantly more complex than MDP environments and due to time limitations of this thesis, it would not have been possible.

2.2. Abstraction

As aforementioned, MDPs can be complex to solve in the ground form, with multiple states and actions leading to a high branching factor. Abstraction offers a way to simplify MDPs by grouping similar states or actions, reducing the dimensionality of the problem while preserving key properties necessary for planning. As such, it is of interest to instead map the ground MDP M to an abstract MDP $\overline{M}$. An abstract MDP is defined as $\overline{M} = \langle \overline{S}, \overline{A}, \overline{P}, \overline{R}, \overline{\gamma} \rangle$, and is created by defining mappings

- $f_s : S \longrightarrow \overline{S}$ which maps ground states s to an abstract state $\overline{s}$

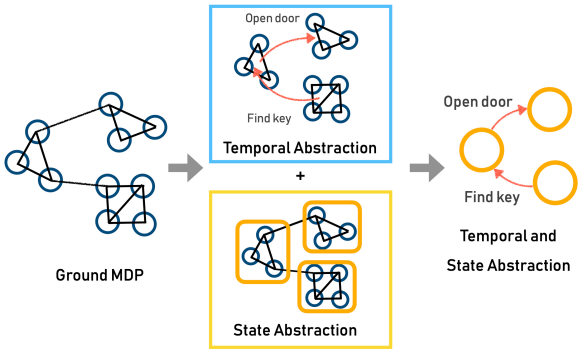


Figure 2.1: A visualization of MDP abstraction [19]

- $g_s : A \longrightarrow \overline{A}$ which maps ground actions a to abstraction actions $\overline{a}$

Similarly, mappings can also be done for P, R and also γ.

As shown in Figure 2.1, the benefit of this is that this can reduce the dimensionality of the ground MDP, which can reduce the number of states and / or actions. This in turn enables solving the abstracted MDP $\overline{M}$ and obtain the abstracted policy $\overline{\pi}$. Using the mapping functions, it is then possible to map the abstracted policy back to the ground MDP such that

$$\pi(s) = g^{-1}(\overline{\pi}(f_s(s)))$$

Therefore it is possible to map the ground MDP to a reduced, abstracted MDP, solve it in the abstracted domain and map the found policy back to the ground MDP. [18] [4] [5] [16].

In Figure 2.1, it can be seen that there are different types of abstraction. When abstracting, a trade-off has to be made based on the objective, relevant information and size constraints. Li et al. determine that there are 5 main ways to perform abstraction, based on the information that needs to be preserved in order to solve the original MDP. The preservation of these 5 pieces of information leads to repetitive abstractions [4]:

- **Model-irrelevance:** if two states have the same immediate reward and transition probabilities for a given action, they are grouped together. It preserves the one-step model.
- Q^{π}**-irrelevance:** if two states have the same action-value function for any actions, they are grouped together. It preserves the state-action value function for all policies.
- Q^{*}**-irrelevance:** if two states share the same optimal-action value for any actions, they are grouped together. It preserves the optimal state-action value function.
- a^{*}**-irrelevance:** states are grouped together based on an optimal action a^{*} that works for both of them. It preserves the optimal action and it's value.
- π^{*}**-irrelevance:** states are grouped together if they both have the same optimal action. It attempts to preserve the optimal action.

In addition to that, Li et al. present a summary of previous abstraction mechanisms used for MDPs. The abstraction methods are listed below in Table 2.1 from coarsest to finest. Coarse abstractions preserve less than finer abstractions, however they are computationally less intense. As such it becomes a trade-off between minimizing information loss and maximizing state-space reduction [4].

2.2.1. Approximate Bisimulation & MDP Homomorphism

From Table 2.1, two methods of abstraction were focused on more closely, specifically approximate bisimulation and homomorphism. These two mechanisms are seen as the most relevant, as they are simple enough to understand and can be performed using intuition, without the need for strict mathematical or statistical means. Furthermore, these mechanisms are used to convey the usefulness and meaning of abstraction to the LLM and as such, covering them is necessary for understanding.

Abstraction Mechanism	Criterion	Exactness	Notes
Bisimulation	Model equivalence	Exact	Strictest measure
Homomorphism	Model equivalence	Exact, matching of actions flexible	Accounts for spatial relations (e.g. symmetry)
Approximate Bisimulation	Model similarity	Bounded	Builds BMDPs
Bisimulation Metrics	Model similarity	Statistically tested	Error bounds deducible
MAXQ	Model equivalence for hierarchically consistent policies	Exact	Integrated into the MAXQ hierarchy
Stochastic Dynamic Programming	Equivalent models given a policy	Exact	Covered by bisimulation
G Algorithm	Equivalent rewards and Q-values	Statistically tested	Each feature's relevance must be independent
Utile distinction	Equivalent best actions with similar Q-values	Statistically tested	May not yield optimal policy for ground MDP
Adaptive Aggregation	Similar Bellman residuals	Bounded	States can be (dis)aggregated dynamically

Table 2.1: Abstraction methods sorted from coarsest to finest [4]

To evaluate abstractions, we draw on the concept of lax bisimulation. Bisimulation traditionally focuses on matching state-action pairs exactly based on rewards and transitions. However, as seen in Table 2.1, this mechanism is the strictest and often does not lead to any or significant reduction. As such, bisimulation was extended to lax bisimulation, which relaxes this notion, allowing 'nearby' behaviors to be treated as equivalent. This makes it suitable for measuring how close an abstraction is to another, where perfect equivalence is often too strict.

$$d_{lax}((s_i, a_i), (s_j, a_j) = c_r \left|R(s_i, a_i) - R(s_j, a_j)\right| + c_t W(P^{a_i}_{s_i s'}, P^{a_j}_{s_j s'})$$

where c_r and c_t are weighting constants, and $W(,)$ is typically the Wasserstein distance (also called Earth Mover's Distance), which compares distributions over next states $P^a_{ss'}$. This in turn lets one match state-action pairs even if they are not exactly equivalent [20] [21].

Leading on, as MDPs often exhibit considerable explicit redundancy, especially when there symmetries. MDP homomorphism exploits these symmetries and derives a smaller, equivalent model for these problems [22]. Figure 2.2 visualizes this and shows that points A and B are simply mirrored in the state space. As such, points A and B are equivalent, like all the mirrored points and therefore one side can be removed to reduce the state space and arrive at an equivalent, abstracted model.

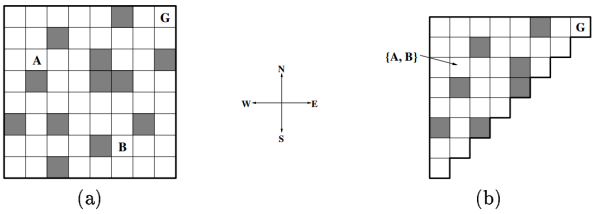


Figure 2.2: A visualization showing MDP homomorphism exploiting the symmetries in the world to reduce the state space [22]

2.2.2. Similarity Metrics

Evaluating abstraction quality is a crucial component of any abstraction-based planning or decision-making framework. In the context of MDPs, it is not sufficient to merely reduce the number of states or actions; An effective abstraction must also preserve the essential behavioral and structural properties of the original model. To this end previous research as conducted to develop similarity metrics, which can generally be divided into structural metrics, behavioral metrics, and performance-based metrics. These allow comparisons between abstracted and original models in terms of fidelity, expressiveness, and task performance.

2.2.2.1. Structural Metrics

Structural metrics are used to quantify how well the dynamics of an abstracted MDP reflect those of the original model. These methods do not rely on learned policies or values but instead evaluate the MDP's internal structure, such as its transition probabilities and reward functions.

One widely adopted structural measure is the Earth Mover's Distance (EMD) (also refered to as the Wasserstein distance above). As previously mentioned, for approximate bisimulation, it provides a principled way to quantify the distance between the next-state distributions of different state-action pairs, accounting for the similarity of their respective outcomes [20].

Another related metric is the Hausdorff distance, which evaluates the maximum deviation between two sets. Applied to clusters of states or actions, the Hausdorff distance captures the worst-case structural mismatch between groups, thus offering an upper-bound measure of dissimilarity. Recent work has adapted this distance for use in abstraction quality metrics, arguing that its metric properties make it useful for comparing coarse and fine partitions of MDPs [20].

Additionally, graph-based metrics have been proposed as a more topological approach to evaluate similarity. Here, MDPs are modeled as directed graphs where nodes represent states and edges encode action-induced transitions. Measures like SimRank compute similarity recursively, under the principle that two nodes are similar if their neighbors are similar. These approaches are especially useful in capturing relational and contextual similarities that may be lost in purely probabilistic comparisons [23] [20].

2.2.2.2. Behavioral Metrics

In contrast to structural metrics, behavioral metrics evaluate the similarity of abstracted and original MDPs based on their resulting behavior. Rather than comparing transition models directly, these methods assess differences in learned policies or value functions.

A standard behavioral metric is the comparison of either state-value functions $V(s)$ or action-value functions $Q(s, a)$. For instance, the Mean Absolute Error (MAE) or Mean Squared Error (MSE) between the values derived from the original and abstracted MDPs can quantify how much abstraction affects the decision quality. These metrics are good when assessing if abstraction-induced simplifications preserve the utility of optimal or near-optimal policies [24].

Another approach to behavioral similarity involves set-based cluster comparisons. For example, Song et al. propose evaluating how well the clusters formed by an abstraction align with those of an ideal or ground truth abstraction. Metrics such as the Adjusted Rand Index or Jaccard Index can be used to measure overlap between sets of grouped states or state-action pairs, penalizing over-clustering and under-clustering accordingly [24].

2.2.2.3. Performance Metrics

Beyond structure and behavior, performance-based metrics provide an empirical means of assessing abstraction quality by executing policies within the abstract and ground environments. These methods evaluate the practical utility of abstractions in decision-making tasks and are particularly relevant when abstractions are used for planning or transfer learning.

One such measure is reuse gain, which assesses the benefit obtained when transferring a learned policy from one task to another using a shared abstraction. Commonly used submetrics include jumpstart (initial performance), asymptotic performance (final performance), and total reward (area under the learning curve) [24]. These indicators help evaluate how well abstractions generalize and how effectively they enable learning in new but related environments.

Another performance-based method involves online similarity estimation, where similarity between tasks or states is inferred during learning itself. This approach allows agents to adaptively decide whether to reuse prior knowledge, improving sample efficiency and avoiding negative transfer.

These different classes of metrics provide complementary perspectives on abstraction quality. Structural metrics assess fidelity to the original MDP's transition dynamics, behavioral metrics evaluate policy preservation and decision alignment. Performance metrics on the other hand, asses the practical benefits in real-world decision-making tasks.

2.3. Monte Carlo Tree Search

As MCTS is the backbone of the evaluation mechanism, it is relevant to consider how it works; MCTS is a decision making algorithm that is used heavily for game applications or decision problems and was famously used in Google Deepmind's AlphaGo [25]. Unlike humans, who rely on intuition, MCTS uses game-tree search combined with Monte Carlo simulations to make strategic decisions. By simulating possible outcomes and accounting for randomness, MCTS can provide a flexible and adaptable strategy over a wide range of games [26].

Di Zhang et al. best summarizes the four main steps of MCTS [8]. Figure 2.3 also highlights these steps visually:

- **Selection:** From the root and based on a specific selection strategy (typically the Upper Confidence Bound) a promising leaf node is selected.
- **Expansion:** At the selected node, one or more new child nodes are added to simulate possible future actions and their resulting state.
- **Simulation:** From the newly created node, the algorithm performs a "rollout", or random simulation of moves until a terminal state is reached. To preserve time, or to avoid simulating for too long, a maximum depth can also be set.
- **Backpropagation:** The outcome of the simulation is propagated back to the root, updating the statistical data (score and visits) of each node traversed to inform about future decisions.

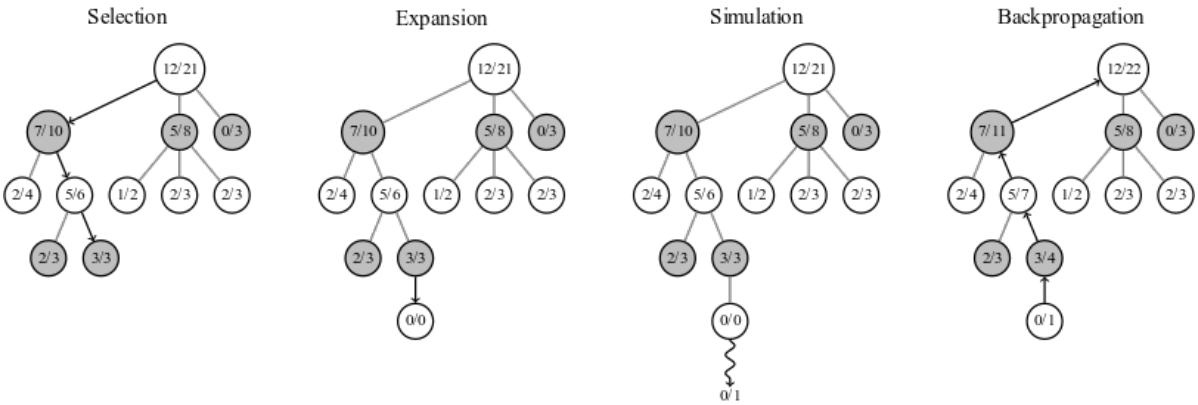


Figure 2.3: A visualization of the 4 main steps in MCTS [27]

2.3.1. Using Abstraction in MCTS

Similarly to pruning abstraction mechanisms can also alleviate the challenges associated with TBS. Abstraction can help MCTS reduce the search space, by clustering states, actions or state-action pairs into equivalent nodes, which simplifies the model for MCTS, while retaining strategic elements [17].

On such approach by Bai et al. sees the use of a hierarchical structure, where groups of states are abstracted into higher level, larger states. For example, in terms of action abstraction, the action `Kill figure A` is simply the composition of `Move next to Figure A` and `Attack figure A`. Hierarchical MCTS extends this to create high-level abstract actions. These options are integrated as nodes in the MCTS search tree, with local policies optimizing decisions within the abstract states. By separating

high-level and low-level decision making, the hierarchical MCTS maintains computational efficiency and is able to make use of the advantages of abstraction [17].

Another notable approach to incorporate abstraction comes in the form of Elastic MCTS (EMCTS). EMCTS extends the theory, by dynamically adjusting the granularity of the abstraction based on the budget. The algorithm starts out exploring ground nodes and after N iterations, the ground nodes are grouped using approximate MDP homomorphism. Search then continues adding more ground nodes until M iterations is reached, when the abstract nodes are split and search continues in all ground states. This leads EMCTS to outperform the baseline MCTS by a large margin with a considerable search tree size reduction [28] [29].

These approaches demonstrate how abstraction can aid MCTS in decision-making to make it a more scalable and effective method for decision making even in complex, high-dimensional environments.

2.4. Large Language Models

Over the course of the last two years in which AI has advanced so much and LLMs have become a staple in out current society, there have been a lot of different applications for them. They have shown to be adept at natural language processing, translation and program repair [12], however their applications have also spread to broader ranges such as coding and math problems [8] [30] as well as games [7].

These broader use cases have been imperative when it comes to showing the range of tasks that LLMs are capable of. Unlike Reinforcement Learning (RL) agents, which often lack the ability to generalize, LLMs are pretrained which gives them extensive world knowledge and the ability to reason over the knowledge. This makes them highly versatile and able to handle complex, real-world scenarios that may involve multiple steps of planning and decision-making [31]. However, it is still unclear though if they exhibit the ability to generate abstract concepts based on only a ”handful” of training samples [11] and when they lack the necessary domain knowledge in specific areas, it leads to irrational decisions [6] and / or hallucinations which are seen as the main limitation [7].

This thesis focused on two classes of LLM Models, the LLaMA family and the Deepseek-R1 family. While the Deepseek-R1 and LLaMA models are both open-source and can be applied to a broad range of NLP tasks, their design philosophies and training methodologies differ. Most notably, the LLaMA models primarily rely on supervised fine-tuning, whereas Deepseek-R1 emphasizes reinforcement learning, particularly for enhancing reasoning capabilities.

From previous benchmarks focusing on reasoning and mathematical problem-solving, Deepseek-r1 models have significantly outperformed their Llama counterparts. For instance, Deepseek-r1 achieved higher scores on the MMLU and MATH-500 benchmarks compared to LLaMA 3.3 [32]. Furthermore, Deepseek-r1 models support a context length of up to 128K tokens, significantly surpassing some of the other LLaMA’s 8K token limit, allowing for better handling of long-context tasks.

2.4.1. Cohesion & Hallucinations

Despite the recent advances, there are still issues that have to be dealt with when working with LLMs. These are object cohesion, how elements in a larger picture work / fit together and hallucinations, generating plausible, but false outputs.

Previous work by Xu et al. studied the Abstract and Reason Corpus (ARC) performance of ChatGPT and found that even on simple ARC tasks it cannot maintain object cohesion over multiple lines. Furthermore, the performance of LLMs deteriorates significantly when objects are not presented. As such the suggested approach is to use object-centric graphs to represent the problem, as it significantly improves the performance of LLMs [11].

Leading on, other papers have reported that hallucinations are the main limitation of LLMs [14] [33], and as such needed strategies to mitigate them. Both approaches by Sun et al. and Madaan et al. use a form of self-refining, in which the LLM answers are scored and improved iteratively to improve the reliability of the outputs [31] [34]. These rewriting techniques were found to mitigate the hallucination issues by making them learn from past experiences (outputs).

2.4.2. LLaMA

The LLaMa family (Large Language Model Meta AI), was chosen for this thesis due to it's open-source nature and suitability for the research purpose. The open-source nature and availability of their weights for download and modification makes models from the LLaMa family ideal for research that requires reproducibility and transparency; This ensures that experiments can be replicated by others without relying on proprietary software [35].

For this thesis, three LLaMa models were chosen specifically:

- **llama3.1:8b** - is a smaller model (8 billion parameters) that should provide baseline performance for tasks while requiring lower computational resources [36].
- **llama3.1:70b** - is a larger model that should display better capabilities, whilst still not using too much computational resources [36].
- **llama3.3:70b** - is the latest model which promises performance similar to llama3.1:450b, but at a fraction of the computational cost and with increased context window, making it a highly efficient and interesting for comparison [37].

Certain models in the LLaMa family were deliberately excluded, this includes the models in the llama3.2 family. The llama3.2 family focuses on models with small parameter count (1B and 3B respectively), and are mainly supposed to be used text summarization or prompt rewriting [38]. Likewise llama3.1:405b, although offering higher performance than the other two models, as can be seen in Figure 2.4, was not considered to be feasible due to the large computational resources required. As practicality and reproducibility were key for this research, it was found not to be suitable.

By selecting these models, this research aims to strike a balance between performance and feasibility. The chosen models align well with the research goals and ensuring scalability and reproducibility for future research. Using open-source models ensures that this research is extensible allowing for any changes in LLM technology.

In summary, the choice of using llama3.1:8b, llama3.1:70b and llama3.3:70b was a deliberate effort to be both open-source accessible, computationally feasible, and high performant. This selection supports the thesis's objectives, while ensuring that the research remains transparent and reproducible.

2.4.3. Deepseek-R1

The Deepseek-R1 model family represents a drastic advancement in open-source LLMs, by emphasizing reasoning capabilities. Developed by the Chinese AI startup DeepSeek [39], these models have garnered attention for their innovative training methodologies and impressive performance benchmarks.

Unlike traditional LLMs that rely heavily on supervised fine-tuning, DeepSeek-R1 models employ a multi-stage training approach centered around RL. The subsequent DeepSeek-R1 model incorporated a ”cold-start” phase, introducing curated chain-of-thought reasoning examples before the RL phase. Architecturally, Deepseek-R1 is based on the DeepSeek-V3 framework, which utilizes a mixture-of-experts (MoE) design. This architecture allows the model to activate a subset of its parameters dynamically, optimizing computational efficiency without compromising performance [40].

DeepSeek has released several distilled versions of the R1 model, including 7B, 8B, 14B, 32B, and 70B parameter models. These distilled models are fine-tuned using data generated by the primary Deepseek-R1 model, ensuring that even the smaller variants retain strong reasoning capabilities [41] [40].

To summarize, the thesis focuses on two types of models Meta's LLaMA and DeepSeek's R1 models. While the Llama models are primarily trained with supervised fine-tuning and selected here for their accessibility and reproducibility, the Deepreek-R1 models emphasize reasoning and employ reinforcement learning during training. DeepSeek-R1 has shown state-of-the-art performance in reasoning benchmarks and supports longer contexts, making it an interesting counterpart to LLaMA models. The combination of both families allows for a extensive and scalable evaluation across tasks central to this thesis.

Category / Benchmark	Llama 3.1 405B	Nemotron 4 340B Instruct	GPT-4 (0125)	GPT-4 Omni	Claude 3.5 Sonnet
General MMLU (0-shot, CoT)	88.6	78.7 (non-CoT)	85.4	**88.7**	88.3
MMLU PRO (5-shot, CoT)	73.3	62.7	64.8	74.0	**77.0**
IFEval	**88.6**	85.1	84.3	85.6	88.0
Code HumanEval (0-shot)	89.0	73.2	86.6	90.2	**92.0**
MBPP EvalPlus (base) (0-shot)	88.6	72.8	83.6	87.8	**90.5**
Math GSM8K (8-shot, CoT)	**96.8**	92.3 (0-shot)	94.2	96.1	96.4 (0-shot)
MATH (0-shot, CoT)	73.8	41.1	64.5	**76.6**	71.1
Reasoning ARC Challenge (0-shot)	**96.9**	94.6	96.4	96.7	96.7
GPQA (0-shot, CoT)	51.1	-	41.4	53.6	**59.4**
Tool use BFCL	88.5	86.5	88.3	80.5	**90.2**
Nexus	**58.7**	-	50.3	56.1	45.7
Long context ZeroSCROLLS/QuALITY	**95.2**	-	**95.2**	90.5	90.5
InfiniteBench/En.MC	**83.4**	-	72.1	82.5	-
NIH/Multi-needle	98.1	-	**100.0**	**100.0**	90.8
Multilingual Multilingual MGSM (0-shot)	**91.6**	-	85.9	90.5	**91.6**

Figure 2.4: Performance capability of llama3.1:405b over different benchmarks as measured by Meta AI [36]

3

Related Work

This chapter surveys prior work that forms the foundation for this thesis, with a specific focus on the use of LLMs in decision-making and planning contexts. LLMs have demonstrated a wide range of abilities, including textual reasoning, environment modeling, and simulating strategic behavior. However, the ability of LLMs to generate structural abstractions remains underexplored. This chapter presents a categorized view of existing approaches, grouped by the roles LLMs play in decision systems, and concludes by identifying the research gap this thesis addresses.

3.1. Prompt Engineering for Reasoning

Prompt engineering is essential for extracting structured reasoning in LLMs, particularly in planning and abstraction tasks where uncontrolled generation can lead to incoherence. Recent literature emphasizes the design of prompts around four key components: instruction, context, input data, and output specification [42].

Instructions: must be specific and directive, explicitly stating the task. Generic or underspecified prompts tend to result in vague outputs or misinterpretations [42] [9].

Context: refers to prior knowledge or task structure necessary for performance. This might include definitions, rules of the environment, or examples of valid abstractions.

Input data: should be preprocessed to match LLM expectations. For instance, tabular representations can be translated into lists or JSON objects describing the contents. Simpler environments like NetHack have shown that success hinges on aligning environment structure with LLM priors [43]. Structured context and inputs, particularly when using JSON or symbolic representations, has been shown to improve cohesion and mitigate hallucinations [11] [44].

Output formats: need to be clearly defined and scoped. Without guidance, LLMs may return unstructured text that is difficult to parse or use. Prompts that enforce response format via templates yield more reliable results and allow downstream parsing and evaluation [42] [34].

Together, these elements constitute an effective prompt composition framework for structured reasoning tasks. They provide the scaffolding needed for LLMs to operate more like structured agents, bridging the gap between textual generation and symbolic abstraction.

3.2. LLMs as Policy Generators

LLMs are often deployed as agents that generate actions directly based on textual prompts. In this role, they act as reactive policies, selecting a next step based on their understanding of the environment encoded in natural language. Examples include base agents in game-like settings, such as the BaseLang agent in CivRealm [15], or agents simulating player behavior in dialogue-heavy games like Werewolf [13]. While these approaches demonstrate that LLMs can select plausible actions, they also expose key limitations, in particular hallucinations and inconsistency in strategy [11] [6].

To address these limitations, prompting strategies such as Chain-of-Thought (CoT) [12], Tree-of-Thought (ToT) [45], and Graph-of-Thought (GoT) [46] have been developed. These techniques aim to scaffold LLM decision-making by decomposing complex problems into manageable sub-steps. CoT encourages the model to think step-by-step, ToT enables backtracking along decision branches, and GoT merges multiple parallel lines of reasoning. A visualization of these strategies can be seen below in Figure 3.1. These methods have been shown to improve coherence and correctness in environments requiring structured planning, something traditionally seen as beyond the scope of LLMs, as they are classified as system 1-type behavior [9] [47].

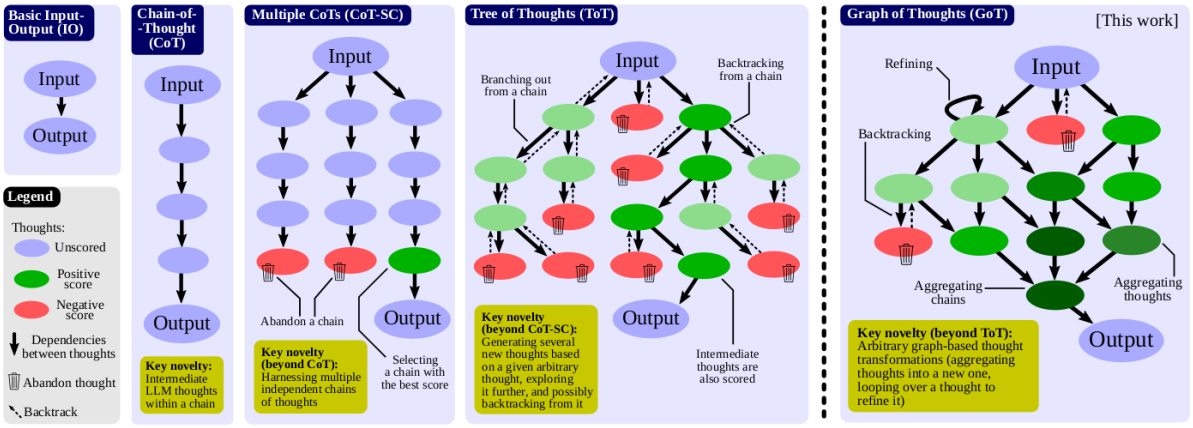


Figure 3.1: A visualization of the different types of prompting strategies that have emerged in previous literature [46]

3.3. LLMs as World Models

Previous research also explores LLMs as implicit models of environment dynamics. In this capacity, LLMs do not merely act, but serve as internal models of how environments evolve. This is particularly useful in scenarios where access to the true transition function is not available or too complex to model symbolically.

Several projects have illustrated the utility of structured input representations for improving LLM performance. Microsoft's NLWeb framework, for example, feeds semantically structured information such as JSON and Schema.org metadata into LLMs to enable coherent natural language responses about websites [44], [48]. These results suggest that environment structure plays a key role in making LLM predictions reliable.

In game-applications and planning environments, this insight translates to modeling the environment with object-centric representations. This is done using textual descriptions, game graphs, or structured plans, before passing them to the LLM [11], [14].

3.4. LLMs as Planners and Simulators

Beyond local decision-making, LLMs have also been used to simulate long-term plans. These approaches often decompose planning into two stages: high-level strategy and low-level action generation. The WarAgent system by Hua et al. is an example, where LLMs simulate the geopolitical reasoning of states during historical conflicts such as World War I and II [33]. To ensure logical consistency, the authors introduce a "secretary agent" that filters and validates outputs before execution.

Hierarchical architectures such as Mastaba in CivRealm [15] follow a similar pattern. A "world observer" LLM generates a strategy, which is then carried out by "worker" agents acting in accordance with it. These hierarchical designs reflect the need to manage decision-making and scale in multi-agent systems. They also highlight the fragility of LLMs when left unrestricted and the utility of separating planning from execution.

Despite these innovations, alignment remains a core challenge. Since LLMs are trained on text rather than domain-specific dynamics, their plans can be incoherent or infeasible unless carefully structured

or validated. Iterative planning methods and feedback loops are commonly used to mitigate this gap [31], [34].

3.5. LLMs in Hybrid Planning Systems (LLM & MCTS)

Recent work has begun to combine the inductive strengths of LLMs with the deductive capabilities of algorithms like MCTS. In hybrid frameworks, LLMs can act as policy priors, heuristics, or simulation engines embedded within the MCTS search process.

In LLM-MCTS [10], Zhao et al. use an LLM to provide a commonsense prior policy to bias the search tree. This allows MCTS to avoid wasting resources on implausible paths. Alternatively, the MCT-Sr framework by Zhang et al. [8] treats the LLM as a node rewriter in a tree of evolving answers. The MCTS process guides which nodes are expanded or refined, improving output quality through search-based iteration.

These systems demonstrate that integrating LLMs into symbolic search can lead to strong performance gains. However, the cost of repeated LLM queries and the complexity of aligning textual reasoning with structured decision processes remain significant issues. Effective integration still requires thoughtful abstractions, prompting, and caching to be viable at scale.

3.6. LLMs for Abstraction and Reasoning

A central question of this thesis is whether LLMs can be used to generate abstractions of structured environments, in this case MDPs. Prior work suggests that LLMs struggle with structural cohesion, especially in visual or logical reasoning tasks that require consistent object tracking or transformation [11].

In the ARC, even state-of-the-art LLMs like ChatGPT fail to maintain object cohesion over multiple steps or scenarios. Performance drops drastically when objects are embedded in unstructured environments. This limitation is echoed in systems like NLWeb, which show that semantic structuring (e.g., JSON or schemas) is essential to enable robust abstraction and generalization.

To date, however, little work has focused on using LLMs to construct abstractions that simplify decision processes. Most prior efforts center on using abstractions as input to LLMs, for example summarizing state-action spaces, rather than extracting them from LLMs themselves. The use of structural reasoning to generate combinations, identify symmetries, or build higher-level representations is still largely unexplored territory.

This thesis seeks to address that gap by evaluating whether LLMs can reliably produce state abstractions and how these abstractions perform under quantitative evaluation.

3.7. Research Gap

Across these different roles LLMs have demonstrated considerable promise in structured decision-making contexts. However, several notable research gaps remain. Firstly, most current systems rely on handcrafted abstractions or manually curated representations. Few research efforts explore whether LLMs can autonomously generate useful abstractions. Secondly, there is a lack of standardized metrics to rigorously evaluate the quality of abstractions produced by LLMs. Existing evaluations are often indirect, relying on downstream task performance or qualitative inspection.

Thirdly, the relationship between LLM characteristics, such as parameter size, architecture, or training data, and their ability to produce meaningful abstractions remains poorly understood. Lastly, aligning LLM outputs with formal decision-making structures, such as MDPs, poses ongoing challenges in terms of grounding and interpretability. This thesis addresses these gaps by proposing a systematic framework for generating, applying, and evaluating LLM-driven abstractions in decision-making environments, contributing both empirical insights and a practical methodology to the broader field of LLM-based planning.

4

Methodology

This chapter covers the methodology of the thesis. It answers two key questions: ”how to extract abstractions from LLMs” and ”how to use abstractions for planning”. Additionally, the metrics used for evaluation are discussed.

4.1. Extracting Abstractions from LLMs

As mentioned in section 2.2 there is a plethora of different ways in which abstraction can be done. In order to stick to the simplest possible method, it was decided to focus on cluster-based abstractions, where states are grouped into equivalence classes. This is in contrast to vector-based clustering which represents states as vectors. The choice to focus on cluster-based abstraction is motivated by two factors. Firstly, clustering aligns with established bisimulation metrics, making evaluation tractable. Secondly, it produces abstractions that can be directly used in planning without requiring additional structural assumptions, such as hierarchies or relational features.

Given this, the goal becomes identifying these state clusters in a way that preserves key behavioral or structural properties of the original environment. The remainder of this section describes the pipeline used to query LLMs for state groupings and extracting valid abstractions.

As outlined in chapter 3 there has been extensive research into how to provide information for LLMs in the best way. However, in order adequately represent an MDP and extract a grouping from the response was one of the challenges of this thesis. To study whether LLMs can generate meaningful abstractions, a prompting pipeline was created that constructs, prompts, and post-processes queries to LLMs. These queries ask the models to group states together based on task-relevant equivalence (e.g., goal reachability, symmetry, or action similarity). The goal of this process is to extract state groupings that are consistent with a homomorphic abstraction of the underlying MDP.

4.1.1. Prompt Construction

The prompt construction process is modular and defined through a composition template. Each prompt is built by combining five main components as outlined in section 3.1:

1. **Instruction:** a directive to the model telling it what to do (e.g., “Group the states into clusters based on similar behavior”).
2. **Necessary context:** a fixed explanation of the underlying rules or abstraction criteria.
3. **Background context(s):** optional prior knowledge or examples to guide the model. It is possible to provide more or less context by using a single or multiple context elements.
4. **Representation:** a way in which the MDP is represented or how data is input to the LLM.
5. **Output format:** explicit instruction to return only a list of lists of integers, representing abstract state groups. Alternatively, it is possible to also tell the LLM to give reasoning. However this can complicate the extraction process.

This prompt structure allows for systematic experimentation with different formulations while maintaining control over the information given to the model. The compositions and raw text fragments are stored and managed in Python, enabling rapid modification and reproducibility.

4.1.2. Querying and Post-Processing LLM Responses

LLMs are queried using the ollama library with a rejection sampling strategy to ensure valid and parsable responses. It is necessary to use rejection sampling, as sometimes the server calls to ollama might fail, models might fail or responses might not be processable. In order to ensure that each model can be fairly assessed, faulty tries are therefore neglected. For each prompt, the following flow follows:

1. The generated prompt is submitted to the selected LLM model.
2. The response is reprompted to the model in order to model self-refinement. This is not a perfect strategy, however it allows the model to rethink the answer and reply with a more refined, and hopefully extractable reply.
3. The reprompted response is inspected. If it contains no usable format, the model is re-prompted with its own output and asked explicitly to extract a list of lists of integers.
4. The reprompted output is cleaned using a regex extractor, which handles diverse formatting (e.g., Markdown, Python lists, cluster labels, braces).

Once the grouping is extracted, it is validated to ensure each state is assigned to exactly one group, no duplicates or invalid indices are included and, if needed, missing states are added back to preserve full coverage. The last point is crucial, as initial experiments had shown that LLMs tend to forget states that don't matter for abstraction, and as such are not included in the final answer. To mitigate this issue, and prevent overusing computational resources, adding missing states was seen as a viable solution as it significantly reduces the amount of retries.

Only cleaned and validated abstractions are kept for evaluation. This filtering process ensures that all abstractions passed to agents are valid from a structural perspective and can be directly used in the simulation. A visualization of the full extraction pipeline can be found below in Figure 4.1, which highlights the different steps, alongside intermediate results to illustrate the need reprompting and cleaning. The image showcases a long prompt (parts omitted for brevity) that gets broken down further upon reprompting, and once cleaned, only the pure cluster-based abstraction is extracted.

4.1.3. Motivation and Benefits

This architecture was designed to be modular and resilient to noise in LLM outputs or errors due to connectivity issues with ollama. By isolating prompt generation, model interaction, and post-processing into distinct components, the system supports rapid iteration, model-agnostic comparisons and automated validation. Furthermore, the approach allows for batch extraction of abstractions, which allows for getting multiple varied replies from a single model across multiple benchmarks.

This pipeline plays a critical role in the experiments as it enables the exploration of the capabilities of LLMs when it comes to producing abstractions. At its core, it was designed to be reproducible and automatable, as it serves as the bridge between the LLMs, and their raw text outputs, and usable cluster-based groupings used for evaluation and planning.

4.2. Using Abstractions from LLMs in Planning

Once valid abstractions have been extracted from the LLM, it is important to examine how these abstractions can be using in planning. This section outlines the methodology for incorporating LLM-generated abstractions into the planning loop, specifically focusing on how these abstractions are incorporated using simulation and decision-making agents.

To ensure modularity and abstraction-agnostic interaction, a stateless simulation backend is used. This backend supports interchangeable simulation modes depending on the type of abstraction provided. In the default ground mode, the simulator operates over the full set of environment states. In contrast, in the abstract mode, the simulator operates over a compressed state space defined by the provided abstraction, mapping between abstract and ground states via a reversible mapping layer.

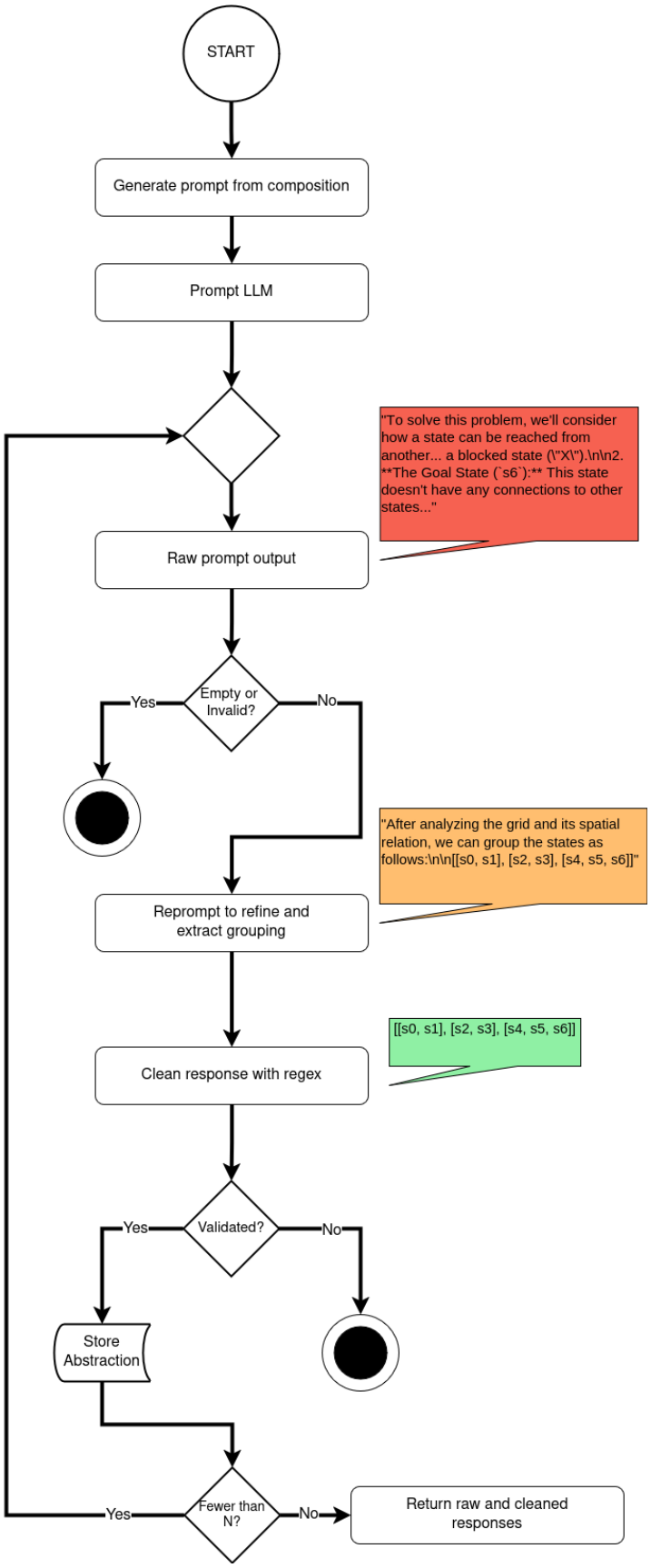


Figure 4.1: Full extraction pipeline showcasing the steps taken and intermediate outputs

The integration of abstractions into planning follows this general pipeline:

1. A set of candidate abstractions is generated and cleaned using the LLM pipeline described in the previous section.
2. Each candidate abstraction is scored using a model-based similarity metric to identify the most promising representation.
3. The top-scoring abstraction is then selected and passed to the simulation framework, which instantiates an abstract simulator by applying the grouping to the full environment model.
4. MCTS agents are initialized using the ground simulator, the ideal abstraction, or the LLM-generated abstraction.
5. The agents run planning rollouts using the configured abstraction level, choosing actions based on forward simulations and search tree expansion.

The simulation layer is implemented in Rust to ensure computational efficiency. It uses a stateless design, where agents do not hold internal memory of the environment, but instead rely on the simulator to advance and evaluate game states. This design supports reproducibility and enables agents to be evaluated under identical conditions across multiple abstraction types.

When an abstraction is passed to the backend, a mapping structure is created, that handles bidirectional translation i.e. between abstract and ground states and actions and vice-versa. The mapping ensures that the agent can reason over abstract states while still interacting with the environment through valid ground-level transitions. Abstract actions are lifted from their representative ground actions and can be mapped but back via a consistent mapping, ensuring coherence. Figure 4.2 visualizes the interaction between MCTS and the abstraction. It highlights how the extracted abstraction is used in the mappings and how state-actions are passed between the agent and the environment.

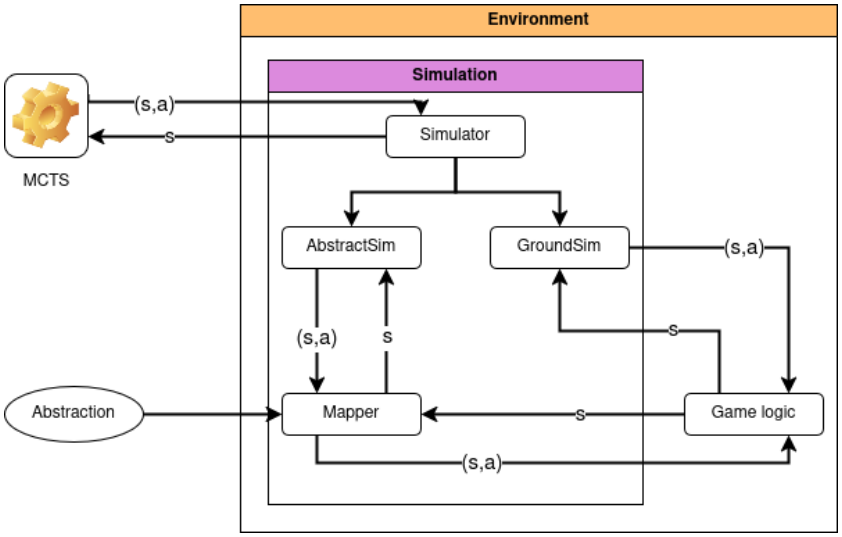


Figure 4.2: Interactions between MCTS, abstraction and the environment

This design allows direct comparison between agents operating with different abstractions under the same planning constraints. The effect of abstraction on decision quality, efficiency, and success rate can therefore be evaluated systematically. More importantly, it demonstrates whether the LLM-generated groupings are not just a ”random clustering”, but also if they are operational decision-making.

4.3. Evaluating Abstractions

In order to effectively compare the abstractions generated by the LLMs, it is important to have a baseline that serves as a comparison. This section covers the model-based and performance-based metrics used for scoring the LLM abstractions. Additionally, this section also makes references to the algorithm used to determine the ideal abstraction for a given map. If interested, the Rust code snippet is provided in Appendix A.

4.3.1. Model-based Metric

To assess the quality of the abstractions generated by the LLMs, a model-based metric, grounded in bisimulation similarity is used to assign a score to each abstraction. This metric helps to evaluate how closely the abstract MDP, generated by the LLMs proposed groupings, matches the ideal abstraction derived from the algorithm and allow investigating how well these abstractions can be used for planning.

State abstraction methods like homomorphism and bisimulation group states that behave similarly under all actions. A proper way to evaluate an abstraction is to compare the generated abstract transition dynamics and reward functions against those of a known ideal abstraction. This idea is formalized through bisimulation metrics, which have been well-studied in RL literature and found to be a reliable indicators of task and policy similarity [24] [20].

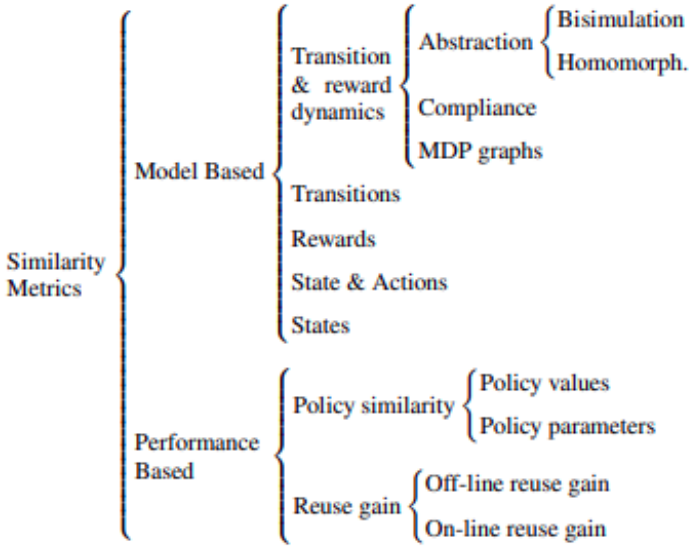


Figure 4.3: An overview of the similarity metrics considered in the survey from Garcia et al. [24]

As shown in Figure 4.3, the scoring method used in this thesis is adapted from García et al. and aligns with a broader class of model-based similarity metrics. This scoring method allows for an immediate evaluation of abstractions, provided the environment's transition and reward models are known.

4.3.1.1. Similarity Metric

Given an abstraction proposed by the LLM and a derived ideal abstraction, it is possible to compute the similarity between the corresponding abstract MDPs. The similarity function is defined as:

$$\texttt{similarity} = \frac{1}{1 + d_M}$$

where d_M is the Hausdorff-style distance that measures the maximal deviation between the abstract states of the LLM-generated and ideal MDPs [49]. This distance is computed using a bisimulation-style metric similar to the lax bisimulation metric as defined in subsection 2.2.1, as it is the weighted combination of reward difference and Wasserstein distance across the abstract transitions.

$$d_M\big((s_i, a_i), (s_j, a_j)\big) = c_r \, |R(s_i, a_i) - R(s_j, a_j)| + c_t \cdot W\big(P(s_i, a_i), P(s_j, a_j)\big)$$

c_r and c_t are weighting constants that trade off reward vs. transition similarity; For all experiments they were both kept at a value of 0.5. This metric decomposes similarity into reward distance, being the absolute difference between the average rewards assigned to each abstract state-action pair, and transition distance, being the Wasserstein distance between the abstract transition distributions. Using the Wasserstein distance $W(\cdot, \cdot)$ is justified as it naturally compares distributions with possibly disjoint

support, allows for comparing transitions across abstract states (which can have different sizes) and produces a single metric.

Additionally, unlike hard correctness checks (for example running exact matching), this metric gives a continuous score in [0,1], where 1 is the same abstraction, allowing meaningful comparisons between partially correct abstractions. In contrast to traditional bisimulation equivalence, which requires exact matching of actions, this scoring function makes use of approximate bisimulation, making it more suitable evaluating partial structures instead of perfect equivalence. The full code snippet for the scoring can be found in Appendix B.

As this metric requires a full transition and reward distributions for comparison, this metric is only computationally feasible on smaller environments. Scalability to larger problems remains an open question. However, as the size of the environment for this thesis is limited, it was found to be a reasonable approach.

4.3.2. Performance-based Metric

In addition to the bisimulation-style model-based evaluation as described in subsection 4.3.1, a performance based metric was also adopted to evaluate how well the abstractions extracted from LLMs perform in practice. The key idea is to treat the abstraction not just as a representational simplification, but to help determine if the abstractions are useful for planning.

MCTS was selected over alternative planning methods such as Value Iteration, Policy Iteration, or Q-learning because it does not require a complete model of the environment's transition and reward dynamics. In contrast to that, MCTS builds a search tree through sampling, making it well-suited for black-box simulators and environments where only a forward model is available. Additionally, MCTS handles large, or abstract state spaces, more gracefully, by adapting its search based on simulation outcomes, rather than requiring exhaustive updates across all states. This aligns with the thesis focus on generated abstraction, where the true structure of the environment is simplified and partial.

Each abstraction is evaluated through gameplay, where an MCTS agent attempts to solve the environment using the given abstraction. The pipeline for this evaluation is as follows:

1. **Prompt the LLM n times** Using the pipeline defined in subsection 4.1.2, n valid groupings are extracted
2. **Score all responses:** Each valid grouping is subsequently assigned a score, using the bisimulation similarity score using the method described in subsection 4.3.1. All n abstractions are assigned a similarity score in $[0, 1]$.
3. **Select best abstraction**: The abstraction with the model-based score is selected for gameplay evaluation, as the highest score indicates the closest similarity to the ideal abstraction.
4. **Run agents in simulation:** Three agents are then evaluated on the same environment using the same MCTS algorithm, each using a different level of abstraction:
    - **Ground Agent** – operates directly on the environment with no abstraction.
    - **Abstract Agent** – uses the precomputed ideal abstraction based on MDP homomorphism.
    - **LLM Agent** – uses the best LLM-generated abstraction from the previous step.
5. **Record performance:** Each agent is run over multiple trials across a sweep of values for the MCTS `simulation_limit` and `simulation_depth`, providing a profile of how well the abstraction supports effective planning under different computational budgets.

While model-based similarity metrics are able to give an insight into the structural fidelity of the abstraction, they do not guarantee that an agent will perform well. An abstraction may have high structural similarity, but may introduce planning blind spots. On the other side, an abstraction with lower bisimulation similarity may still support effective decision-making if it compresses the state space while preserving the relevant transition and reward dynamics.

Therefore, the performance-based metric measures actual agent behavior and helps identify whether the abstraction helps the agent reach the goal more efficiently. By comparing the score, amount of

turns taken, and goal completion across the three agents, it is possible to quantify the practical utility of the abstraction.

These results provide a functional benchmark that complements the model-based score and help establish whether LLM-generated abstractions can be used effectively in real-world planning agents as per the sub-research questions.

4.3.3. Combined Metric: Structure and Planning

While the model-based and performance-based metrics measure a different aspect of abstraction quality, neither fully explains whether an LLM-generated abstraction is both structurally and practically useful. For example, an abstraction may score poorly on structure, yet still yield strong planning outcomes, or vice versa. Therefore, to allow for a full comparison of models, prompts and model-prompt configurations, this thesis also introduces a combined evaluation metric that integrates both perspectives into a single composite score, aimed to balance these perspectives.

The core idea behind this combined metric is to balance structural fidelity (how close the abstraction is to the ideal structure) with planning utility (how well the abstraction performs in simulation). This can be done by considering the model-based score and the performance deviation, which measures how much the MCTS agent using the LLM-generated abstraction, lags behind the MCTS agent using the ideal abstraction.

To ensure that neither metric outweighs the other, they are standardized using z-scores, which indicates how far above or below the mean value a specific value is.

$$z = \frac{x - \mu}{\sigma}$$

Where z_s reflects the relative model-based efficacy and z_g reflects the relative performance-based efficacy. z_g is standardized such that higher values mean worse performance, hence subtracting it aligns the directionality. As such, the final composite score can be calculated as:

$$z = z_s - z_g$$

By aggregating the scores across all maps, models, prompts and / or model-prompt combinations, this metric provides a global view of abstraction performance that accounts for both correctness and utility. It also highlights generalization across environments of different sizes and abstractability levels.

This combined evaluation is important, as correctness alone is insufficient; An extracted abstraction may look valid and receive a high model-based score, but may fail in planning due to faulty symmetries or transitions. On the other hand, an abstraction that deviates more from the ideal may still support effective planning. Therefore, combining both metrics is essential to answer this thesis' research questions about whether LLMs produce abstractions that are not only structurally valid but also usable by real planning systems.

5

Experiments

This chapter covers all relevant information regarding the experiments. It covers the experimental design, such as the environment and the flow of the experiments. From there, each sub research question is detailed again, going over the questions, how they impact the experiments, setup and lastly results to answer them.

5.1. Experimental Setup

This section outlines the shared experimental setup used throughout the study. It includes the details of the environment, agent configurations, abstraction evaluation pipeline, and simulation parameters. The setup was designed to ensure controlled, repeatable experiments across different prompts, LLMs, and map complexities.

5.1.1. Environment

In the code itself, the environment logic and simulation is written in Rust. This was done as Rust is fast and memory-efficient. Additionally, it can easily integrate with other languages, such as Python which is required to make calls to LLMs using ollama. Furthermore, Rust's rich type system and ownership model guarantee memory-safety and thread-safety, which eliminates many classes of bugs at compile-time, unlike with C or C++ [50].

The environment in which the LLMs and agents will interact with is a simplified grid-based environment resembling a maze. It is based on the basic grid world examples outlined by Sutton and Barto. A visual reference for this game can be found below in Figure 5.1. The environment is deterministic and fully observable, and the agent must navigate from a start state (always located in the top-left corner) to a goal state (always located in the bottom-right corner). In the ground environment, the agent has the possibility to choose from 4 possible actions: `up`, `down`, `left` and `right`. In case the agent is at the edge of the field, or tries to move into a cell that is blocked by an obstacle, the agent's location will remain unchanged [51]. The agent receives a reward of `+1`, by moving onto the tile marked as goal and the episode ends.

Unlike in the game that Sutton and Barto present, in this environment, the agent does not receive a reward of `-1` if trying to move outside of bounds or into an obstacle. This simplifies the scenario further helps the MCTS algorithm clearly determine long-term strategies without too much computation. In addition to that, the maps are restricted to be strictly square size, as this simplifies calculating the exact homomorphic abstraction.

To vary size and complexity, maps of three sizes were used: 3×3, 5×5, and 9×9, each with multiple structural variants. These variants, which in the context of this thesis are referred to as *abstractability*, is a category used to determine how far the state space can be compressed, while preserving the underlying MDP. Abstractability is assigned by computing a reduction factor R, which is found by comparing the total number of ground states n with the number of abstract states k in the ideal abstraction.

[h]

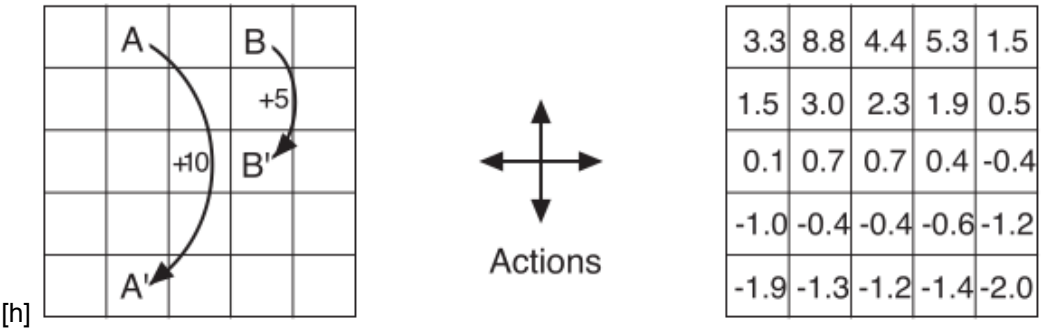


Figure 5.1: The grid world environment as outlined by Sutton and Barto [51]

$$R = \frac{n-k}{n}$$

Based on the reduction factor, the generated maps can be grouped into the following categories:

- **No abstraction maps:** no symmetries or symmetrical structure ($R = 0$).
- **Partial abstraction maps:** contain local symmetries or regularities ($0 < R < 0.25$).
- **Perfect abstraction maps:** exhibit global symmetries or structure ideal for compression to reduce the state space significantly ($R \geq 0.25$).

Each map defines a different MDP, and their hash identifiers are used to ensure reproducibility across runs. The ground truth abstraction (used as an ideal reference) is computed using a homomorphism-based Rust implementation.

5.1.2. Shared Pipeline of Experiments

The previous chapter chapter 4 outlined the overall methodology and motivation of the experiments conducted in this thesis, while subsection 5.1.1 discussed the environment that is shared among all the experiments. This pipeline used across all experiments to answer the research questions as well as the subresearch questions. A visualization can be found below in Figure 5.2.

In its entirety, the flow for evaluating a prompt, on a model on a map can be broken down into the following main steps:

1. Generate the environment map, representations and prompt composition in order to build the prompt.
2. Using the prompting and prompt processing pipeline described in subsection 4.1.2, extract n valid cluster-based abstractions.
3. Score the groupings using the bisimulation similarity metric as outlined in subsection 4.3.1.
4. Once all groupings have been extracted, validated and scored, select the best abstraction (i.e. with the highest score).
5. Initialize the runner to evaluate 3 MCTS agents as outlined in subsection 4.3.2.
6. Log the responses, extracted groupings and MCTS results.

In the empirical evaluations, the number of valid abstractions extracted per configuration was fixed at $n = 20$. This value was chosen to balance computational feasibility with the need for investigating variability in the generated outputs. Extracting 20 abstractions per experiment provides a sufficiently diverse sample to capture the variance in model behavior, while remaining tractable in terms of runtime and resource usage.

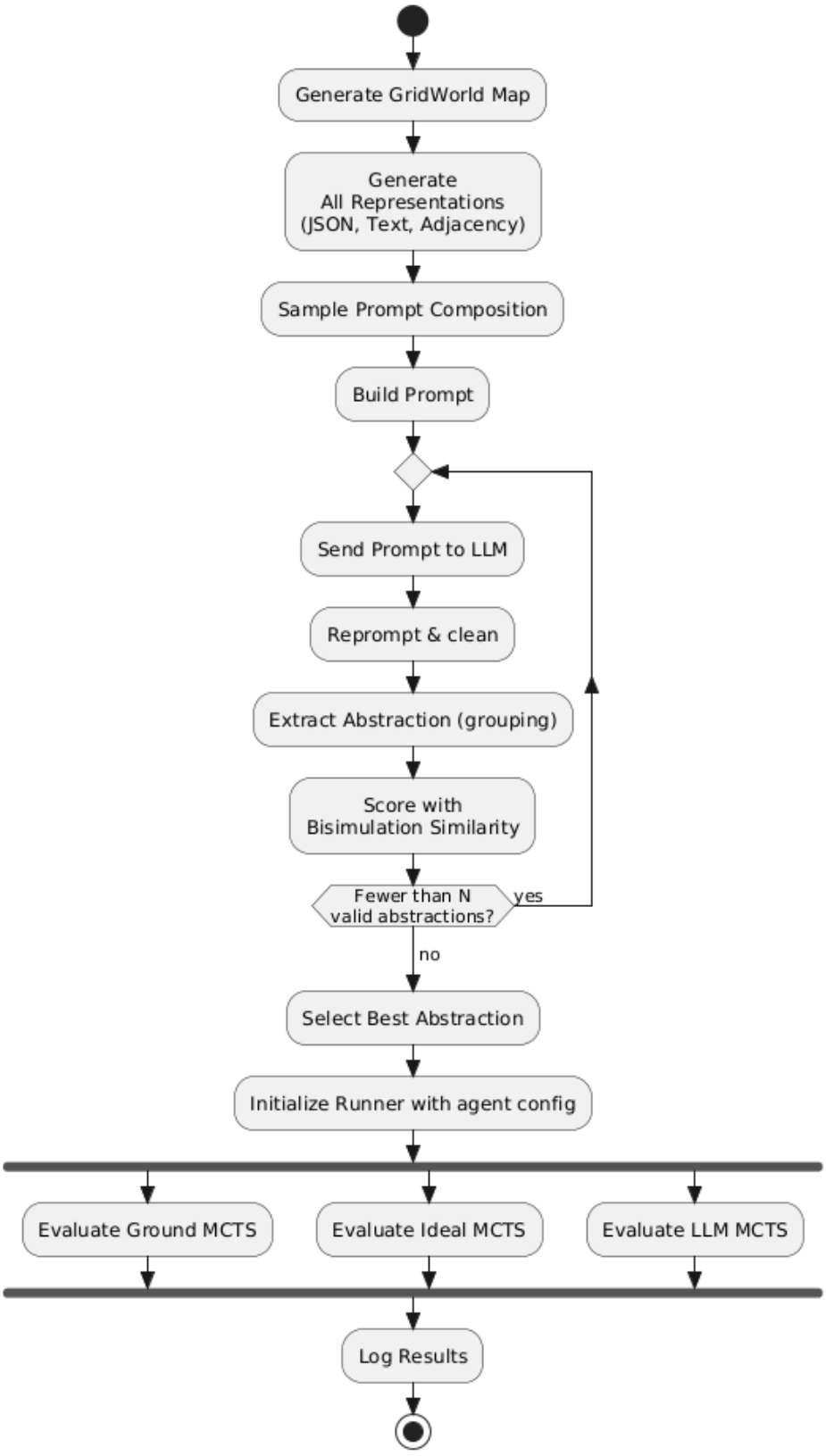


Figure 5.2: The experimental flow applied when testing an LLM model for a prompt on a map

5.2. Experimental Design and Results

This section presents the core experimental findings of the thesis and is organized to systematically address the main research question: *Can LLMs be used for abstractions in MDPs?* Building on the shared setup outlined previously, it is now possible to dive into how the experiments were designed, in order to evaluate the quality and utility of LLM-generated abstractions. Two primary evaluation perspectives guide this investigation:

- whether LLMs can approximate ideal abstractions when available (model-based evaluation)
- whether these abstractions support effective planning using MCTS (performance-based evaluation)

The experiments are further structured around three key factors hypothesized to influence abstraction quality: the choices of LLM, the compositions of the prompt, and the complexity of the environment. Each of these dimensions is examined through a dedicated subsection, in which the relevant sub-research questions are revisited, followed by a description of the experimental setup and a presentation of the results.

5.2.1. LLMs

LLMs vary significantly in terms of architecture, training data, instruction tuning, and scale. These differences can influence their ability to generalize over environments, maintain structural coherence, and produce valid abstractions. Given the growing diversity of available LLMs, it is essential to investigate whether certain models are inherently more suited, or better performing, for abstraction tasks than others.

This motivates the first sub-research question in this category:

How does LLM model type and size affect the quality of the abstractions they produce?

Understanding this relationship helps identify whether LLM type or size correlates with abstraction performance, and whether general-purpose instruction-tuned models can serve as reliable abstraction mechanisms in decision-making contexts. Additionally, this question offers insight into how transferable LLMs are as agents in structured domains such as MDPs, particularly when operating under minimal domain-specific training.

5.2.1.1. Setup

The LLM models used in this study were previously introduced in section 2.4 and include the following variants. The number following each model name indicates the (approximate) number of parameters; The suffix `b` signifies billions:

- llama3.1:8b
- llama3.1:70b
- llama3.3:70b
- deepseek-r1:7b
- deepseek-r1:8b
- deepseek-r1:14b
- deepseek-r1:32b
- deepseek-r1:70b

These models were selected based on three criteria. Firstly, they are open-source and can be run locally via the Ollama framework. Secondly, they span a range of model sizes to examine scaling effects, and lastly, they include recent variants from distinct model families (e.g., LLaMA vs. DeepSeek) to assess architectural diversity.

All experiments were run on the DelftBlue supercomputing infrastructure. The available GPU nodes (`gpu-a100`) are equipped with 4 × NVIDIA A100 GPUs, each with 80 GB of VRAM. Due to these hardware constraints, the maximum model size used was 70 billion parameters, which fits comfortably on a single node without model parallelism. Larger, more well-known LLMs, such as GPT or Claude, were

not considered due to the lack of public access or the infeasibility of local inference without proprietary APIs [52] [53] [54].

Each model was queried 20 times per combination of map and prompt, using the same prompt structure across models. This ensured that any differences in abstraction quality could be attributed to the model characteristics rather than prompt design. The evaluation pipeline, detailed in section 5.1, scored each abstraction using the model-based metric and selected the best-scoring abstraction per configuration for performance-based evaluation using MCTS.

To maintain experimental consistency, temperature, top-p, and repetition penalties were kept at default values provided by the Ollama backend. The same set of maps and prompts was used across all model evaluations. MCTS parameters of $c = 1.4$ and $\gamma = 0.85$ were used across all experiments.

5.2.1.2. Results

The performance of each LLM was assessed through two complementary evaluation metrics mentioned above: the model-based metric, and a performance-based metric. These were later combined into a unified ranking using a composite z-score to identify models that performed well both structurally and functionally.

As shown in Figure 5.3, which shows the model-based score distribution per LLM model, they produce a wide range of scores. All models, with exception of llama3.1:70b are at some point able to achieve the exact abstraction. However, the overall mean scores are low across all models (around 0.2 - 0.3) which indicates that extracting the exact abstraction is rare.

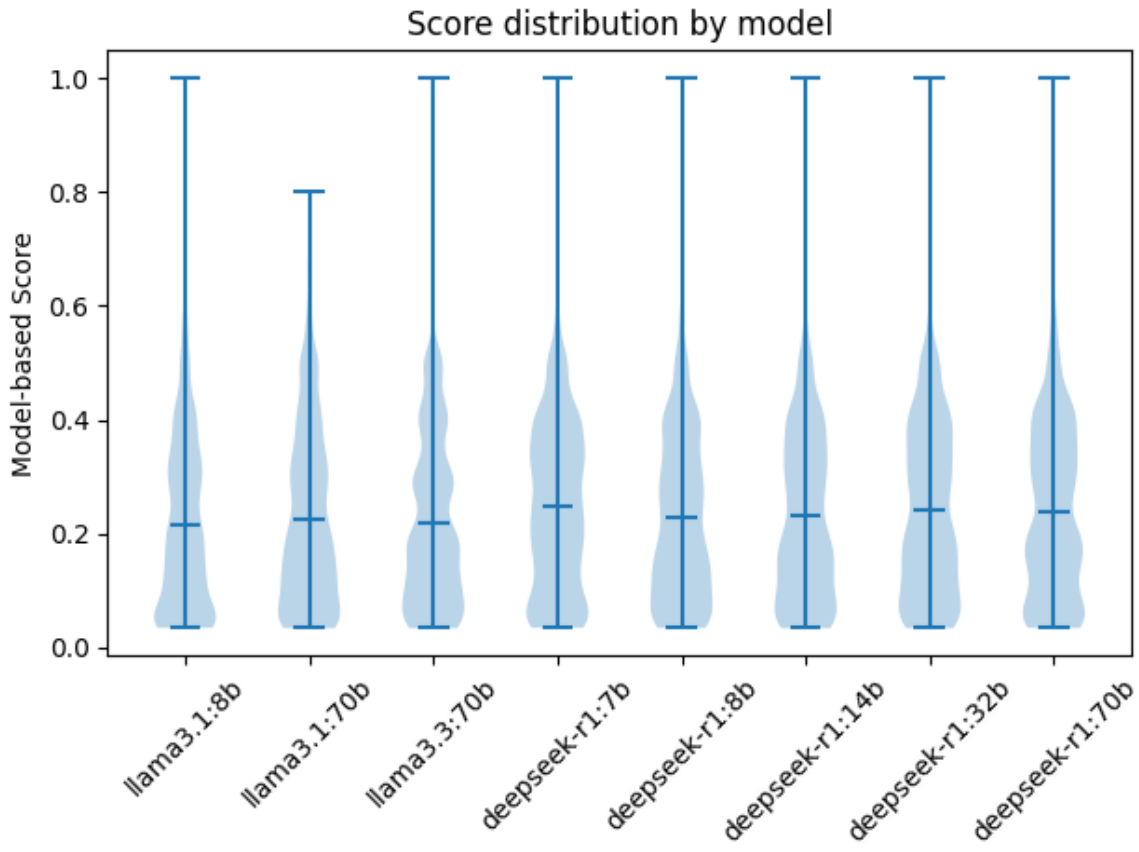


Figure 5.3: Violin plot of the model-based score distribution per LLM model

The Deepseek-r1 models, show a higher median and upper quartile in model-based scores, showing that overall they perform better. To further, analyze the effect of the models and sizes, an ANOVA analysis was performed which was then tested using Tukey's Honestly Significant Difference (HSD) across all

maps, to identified the top-performing LLMs. Across all maps, the leading models are deepseek-r1:7b, deepseek-r1:32b, deepseek-r1:70b, with deepseek-r1:7b consistently outperforming others. This is emphasized by Figure 5.4, showing that, on average, the 7b deepseek-r1 LLM scores better in the structural metric.

It is to be notes though, that the structural scoring is not the sole indicator for performance. If the model just simply returns each state as a cluster, it is a valid grouping, despite it not reducing the space in any meaningful way. In turn, it also scores higher in the structural metric and therefore consistently achieves a higher score.

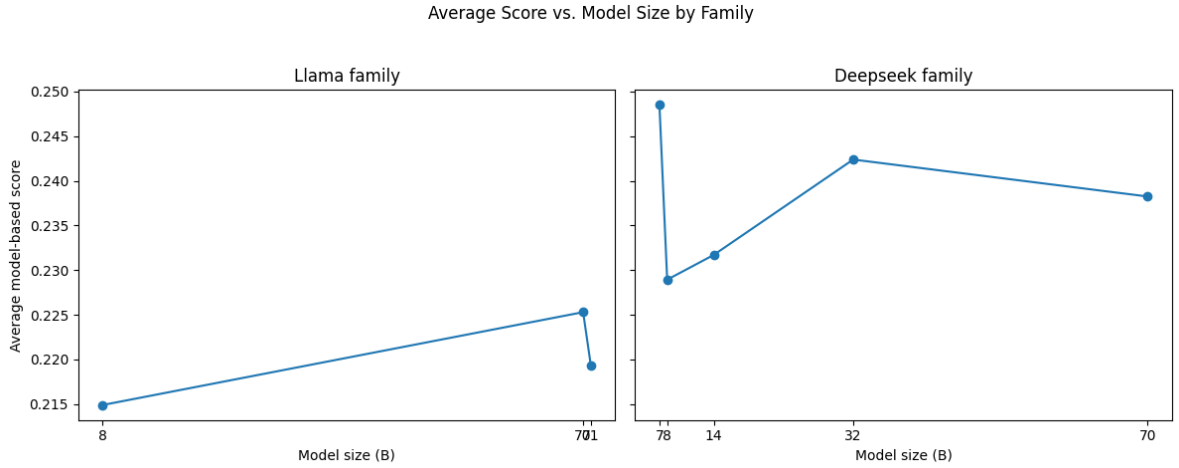


Figure 5.4: Line plots showing the model-based score differences across the families and model-sizes. llama3.3:70b is denoted using a size of 71 to differentiate the models on the graph.

Some models are able to achieve equivalent or even better planning performance with the extracted abstractions. This can be seen in the figures in Appendix E, which shows the deepseek-r1:7b model performing equivalent to the perfect abstraction and even outperforming it. Deepseek-r1:7b is not the only model that achieves such performance, however with increasing map size, the performance significantly deteriorates. Furthermore, planning performance was not always correlated with model-based score. Some abstractions with low structural similarity scores still produced good planning outcomes, and vice versa.

Before drawing conclusions, it is also of interest to examine the composite z-score, as it doesn't just focus on a single metric, but allows for comparison across both. Table 5.1 shows the overall ranking and Figure 5.5 visualizes it as a point plot. Both indicate that the Deepseek R1 LLMs are superior to the llama LLMs, and surprisingly, the most recent model of the llama family, llama3.3 actually performs the worst.

Model	Composite z-score
deepseek-r1:32b	0.3441
deepseek-r1:7b	0.2318
deepseek-r1:70b	0.1644
deepseek-r1:8b	0.1201
deepseek-r1:14b	0.1030
llama3.1:70b	-0.0807
llama3.1:8b	-0.2308
llama3.3:70b	-0.5473

Table 5.1: Ranking from best to worst of the models using the composite z score, across all maps and prompts

These rankings confirm that model size and architecture both influence abstraction ability, but not always linearly. We can see that deepseek-r1 models clearly outperform the llama family, which was to

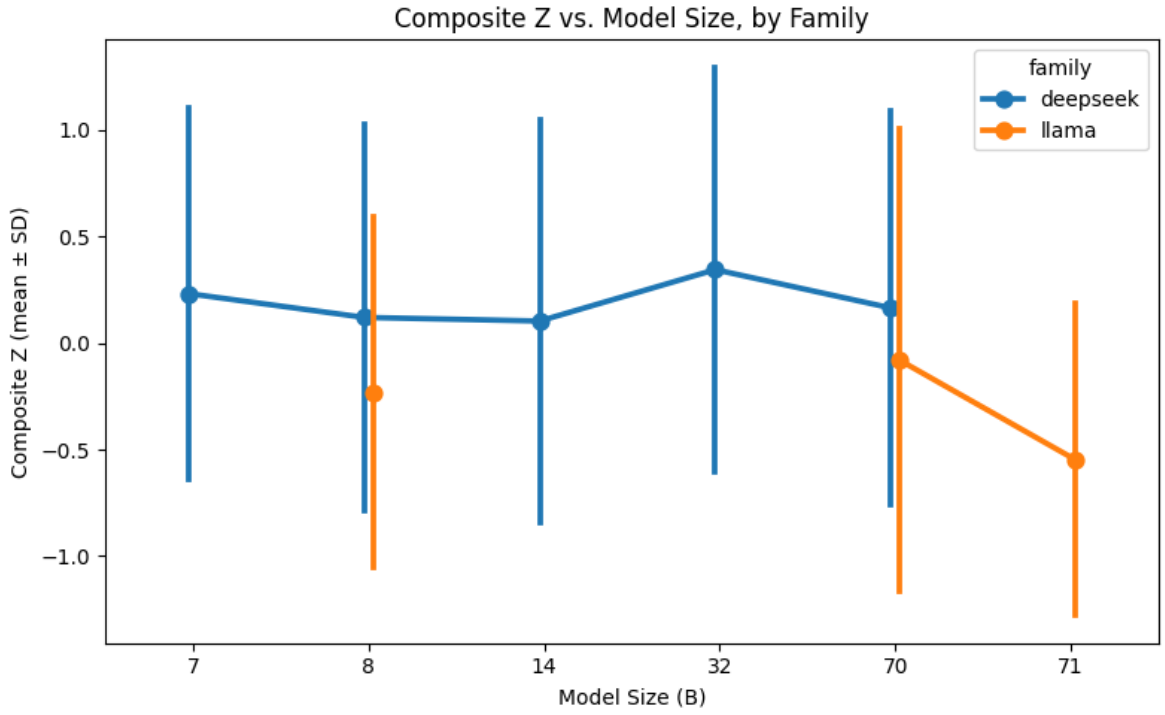


Figure 5.5: Point plots of the average performance of the LLMs across families and sizes, using the composite-z score

be expected. Moreover, the composite metric provides a more robust evaluation than either structural or behavioral metrics alone, and highlights cases where models can produce useful abstractions even when not strictly optimal.

5.2.2. Prompts

LLMs are inherently prompt-sensitive and their performance can vary significantly based on how tasks are described, structured, and represented. Since abstractions are a form of structural reasoning, the way an environment and task are conveyed to a LLM may significantly influence its ability to induce meaningful partitions. Furthermore, the ambiguity or verbosity in prompts may lead to hallucinated states, inconsistent outputs or misunderstood dynamics, particularly in symbolic or spatial domains like MDPs. This motivates the first sub-research question in this category:

How does the phrasing and composition of prompts affect the quality of abstractions?

Additionally, since LLMs are not trained to operate on low-level spatial inputs, the way that information is passed is also of significance. Previous literature already suggests that structured input performs better. Therefore we also ask:

Is there a way to represent the environment that is most optimal for LLMs?

Understanding the influence of prompt structure can inform future work on designing robust language interfaces for structured tasks, particularly in domains like planning.

5.2.2.1. Setup

Before running the main experiments, a prompt selection phase was conducted to reduce the design space and identify promising prompt structures. In this phase, a large number of prompt compositions were systematically generated by varying five key components:

- **Instruction** – how the task is framed (e.g., role-based vs. relation-based phrasing),
- **Necessary context** – core information about the task and abstraction definition,
- **Background context(s)** – optional examples or domain-specific details,
- **Representation** – the format used to describe the environment (e.g., text, JSON, or adjacency matrix),
- **Output format** – specification of what format the LLM should return (e.g., raw JSON cluster list).

All candidate compositions were built using prior literature, online prompt collections, and iterative refinement using ChatGPT. A full list of the initial prompt variants is provided in Appendix C.

Similarly to before, temperature, top-p, and repetition penalties were kept at default values provided by the Ollama backend. The same map and models were used for all evaluations.

5.2.2.2. Results: Prompt Selection Phase

The results from the initial prompt tuning are summarized in Table 5.2, which lists the top 10 performing prompt configurations based on a preliminary version of the model-based metric, which produced a non-normalized loss score; Lower scores correspond to more desirable abstractions. Each prompt was tested on a small, unobstructed 3×3 map and tested on three different LLM variants: llama3.1:8b, llama3.1:70b, and llama3.3:70b. These results informed the final prompt choices for the core experiments. Prompts using adjacency matrix representations were consistently outperformed and thus excluded from the main experiment design.

Rank	Instruction	Necessary Context	Context	Output	Representation	Avg MB Score	Avg Error Rate
1	role1	necessary-domain2	domain2	out2	text	5.2049	0.0000
2	relation1	necessary-domain2	test3	out2	json	5.6105	0.1583
3	role3	necessary-domain2	domain2	out2	text	5.6331	0.0750
4	role1	necessary-domain2	domain2	out2	json	5.7324	0.1167
5	gpt2	necessary-domain2	background1	out4	json	5.7851	0.2733
6	relation1	necessary-domain1	test2	out5	json	5.8256	0.1529
7	role1	necessary-domain1	test3	out1	json	6.0676	0.1714
8	role3	necessary-domain1	domain3	out2	text	6.1205	0.1500
9	gpt2	necessary-domain2	background1	out4	adjacency	6.1528	0.1286
10	relation1	necessary-domain1	test2	out5	text	6.1632	0.1111

Table 5.2: Top 10 LLM prompt compositions by average score and error rate.

From this, a selection was made based on how well different elements perform, overall composition scores and error rate and 5 new compositions were made. Both text and JSON representations were

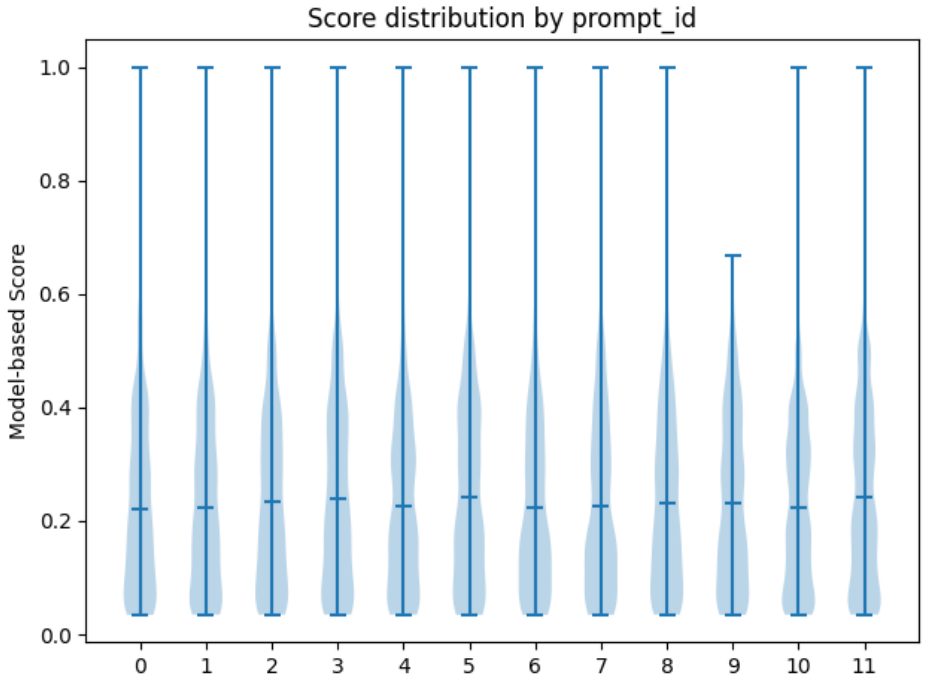


Figure 5.6: Violin plot of the model-based score distribution per prompt index

retained. One additional composition, created through exploratory testing using ChatGPT, was also included due to its robust performance across models. In total, 12 final prompt variants were selected for use in the main experiments (see Table 5.3).

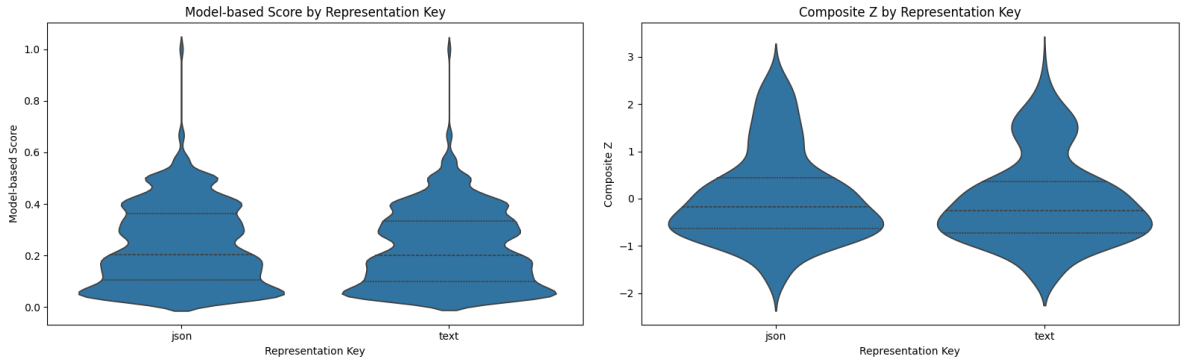
#	Instruction	Necessary Context	Background Context(s)	Output	Representation
1	role1	necessary-domain2	domain2	out2	text
2	role1	necessary-domain2	domain2	out2	json
3	role1	necessary-domain1	test3	out1	text
4	role1	necessary-domain1	test3	out1	json
5	relation1	necessary-domain2	test3	out2	text
6	relation1	necessary-domain2	test3	out2	json
7	relation2	necessary-domain2	performance3	out2	text
8	relation2	necessary-domain2	performance3	out2	json
9	role3	necessary-domain1	domain3	out2	text
10	role3	necessary-domain1	domain3	out2	json
11	gpt1	domain2	test1, test3, assumption	out4	text
12	gpt1	domain2	test1, test3, assumption	out4	json

Table 5.3: Final prompt compositions used in experiments. Includes top-performing prompts and one generated by ChatGPT.

5.2.2.3. Results

Across all models and environments, abstraction quality was highly sensitive to prompt phrasing and structure. As shown in the prompt ID violin plot (Figure 5.6), while most prompts were able to occasionally produce high-scoring abstractions, the distribution of scores was heavily skewed, with the majority of outputs clustered around 0.2–0.3. This highlights the importance of prompt design and the inherent variance in LLM outputs.

Prompts using the JSON format outperformed those using plain text, both in model-based scores and in composite z-scores. This supports prior findings that structured inputs help LLMs maintain cohesion and avoid misrepresentation of the domain. The JSON representation likely improves alignment with the LLM's pretraining on structured data formats, leading to more consistent clustering behavior (see



(a) Model-based metric

(b) Composite-z metric

Figure 5.7: Violin plots of the score distributions for the model-based metric and the composite-z metric, highlighted for the representation

Figure 5.7).

This is further supported by Figure 5.8, which shows that llama models significantly suffer in performance when using a text-based representation. Deepseek-r1 models also experience a drop in performance, however not as strong. Lastly, the only model that experiences an improvement from a text-based representation is deepseek-r1:70b. This might suggest that structured JSON input likely serves as an inductive prior, especially for smaller or supervised models like llama. However, larger or reinforcement-trained models may be more capable of inferring structure from free-form text, which may even benefit from its expressive flexibility.

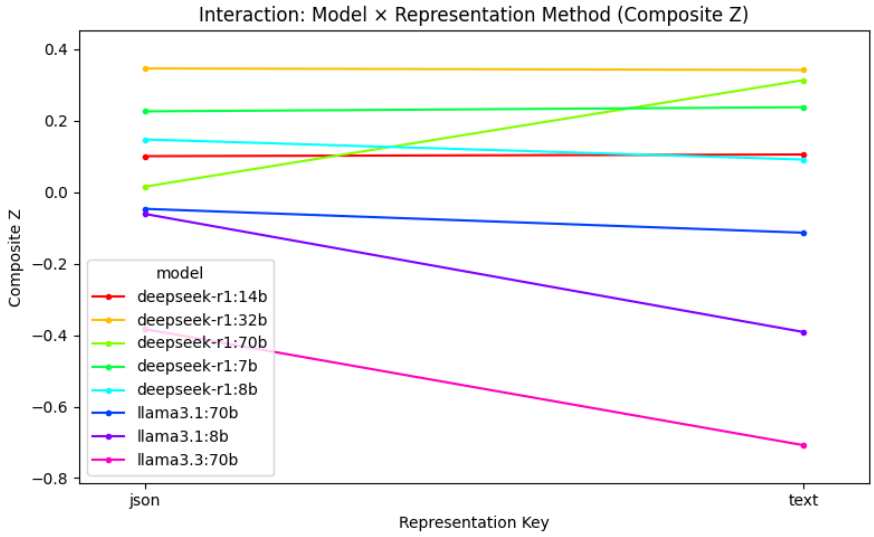


Figure 5.8: Interaction graph of LLM models and representation key using the composite-z metric

To understand which components of a prompt contribute to better abstraction quality, the final experimental data was analyzed both via raw model-based scores and a composite z ranking score. Each of the five prompt components was analyzed using Tukey's HSD test to compare all pairwise group differences in the composite score space.

Instruction phrasing had a notable impact. The prompt category `gpt1`, which was derived from exploratory prompt development using ChatGPT, emerged as the highest-performing instruction according to the Tukey's HSD. `relation1` and `relation2` were statistically indistinguishable from `gpt1` but showed slightly lower means. Further analysis needs to be conducted as the prompts with `gpt1` include more context, which could also lead to improvement.

For the necessary context, the analysis confirmed that `domain2` was statistically superior to both `necessary-domain1` and `necessary-domain2`, with a significant mean difference. In terms of background context, the top-performing variants were `domain3` and `domain2`.

Output format also showed measurable effects. Prompts labeled `out4` consistently ranked higher. This output format asked for an additional rationale for the state groupings, alongside the answer, indicating that forcing the LLM to provide reasoning, may in-fact help to derive better conclusions and therefore results.

Across all components, the analysis confirms that several prompt design choices have statistically reliable effects on abstraction quality. These findings reinforce the idea that LLM performance is highly prompt-sensitive, and that careful prompt engineering, including specific language templates, domain-aware context, structured representation, and clear output formatting, can significantly enhance results.

5.2.3. Maps

The final concern of this thesis is the environment. Specifically, whether the complexity and structure of the environment, as reflected in different map layouts, influence the quality of LLM-generated abstractions. This is motivated by two core sub-research questions:

> *How does the quality of abstractions change as the complexity of the environment increases?*
>
> *What happens when no exact abstraction exists — can LLMs still find useful compressions of the environment?*

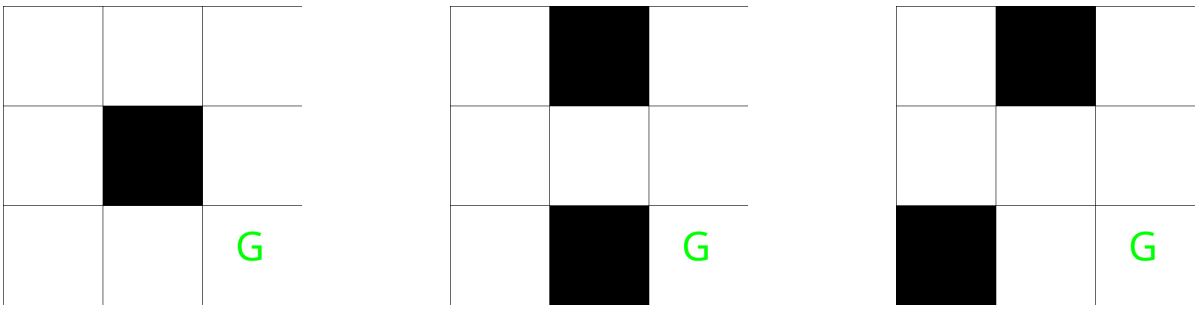
As environments become larger or less structured, the set of valid or useful abstractions becomes increasingly sparse or noisy. This presents a challenge to LLMs, especially when operating under limited task descriptions. Conversely, highly structured environments may induce favorable biases that allow even small LLMs to detect global symmetries. These questions aim to understand how sensitive the abstraction process is to the environment's structure, and whether LLMs can still produce meaningful simplifications when ideal abstractions do not exist.

5.2.3.1. Setup

As described in subsection 5.1.1, maps used in the experiments are deterministic grid worlds of size 3×3, 5×5, and 9×9. Each size is paired with three manually constructed variations that differ in their abstractability. Each categorization, defined by the reduction factor R, serves a specific role in the evaluation:

- Perfect abstraction maps contain global symmetries and compressible dynamics ($R \geq 0.25$). These maps validate the LLM's ability to reproduce ideal abstractions.
- Partial abstraction maps include local or partial symmetries ($0 < R < 0.25$) and assess robustness in ambiguous settings.
- No abstraction maps offer no structural regularity ($R = 0$). These maps probe how LLMs behave when no valid compression exists.

The full list of maps used, including visualizations, is shown in Figure 5.9, Figure 5.10, and Figure 5.11. These maps are reused across all prompt and model configurations, ensuring controlled comparisons across experimental axes.



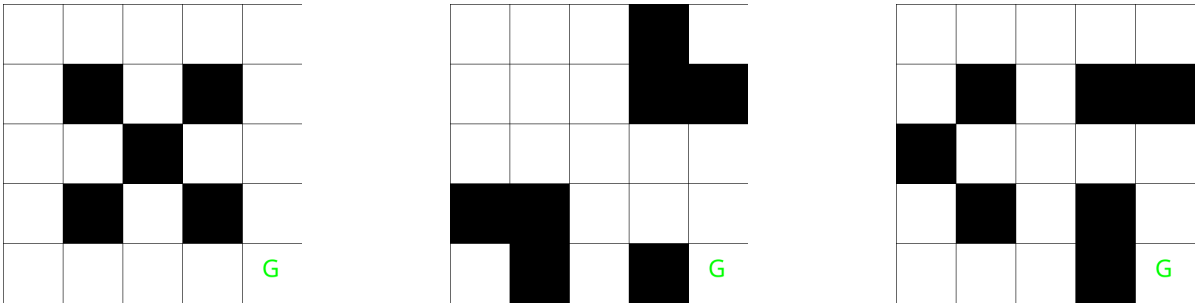
(a) Perfect abstraction **(b)** Partial abstraction **(c)** No abstraction

Figure 5.9: Maps of size 3 by 3 showing varying levels of abstractability.

5.2.3.2. Results

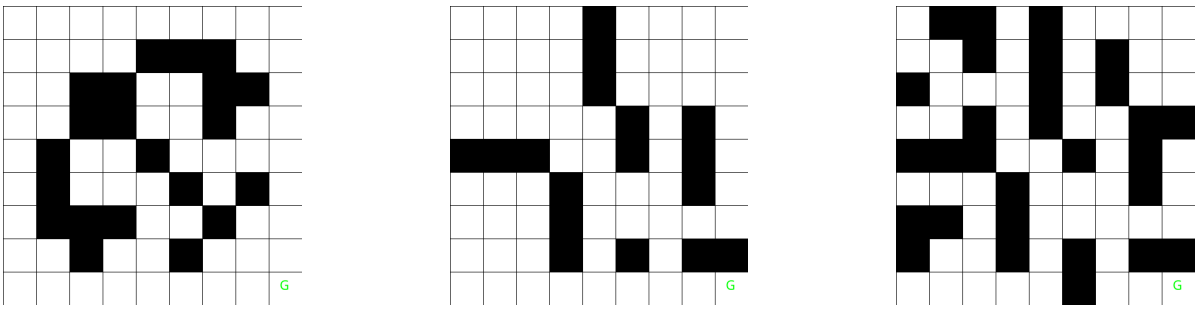
The results demonstrate a consistent relationship between environmental complexity (as captured by map size and abstractability) and the quality of LLM-generated abstractions. Two overarching patterns emerge from the analysis.

Firstly, increasing the size of the environment leads to a clear decline in abstraction quality. As shown in Figure 5.12, the average model-based score decreases across all levels of abstractability as the map size increases from 3×3 to 9×9. On small 3×3 maps, LLMs are capable of identifying meaningful groupings, even in cases with no clear underlying structure. However, by the time the size increases to 9×9, performance deteriorates significantly, especially for no-abstraction and partial-abstraction maps. This trend is also observed when using the composite z-score.



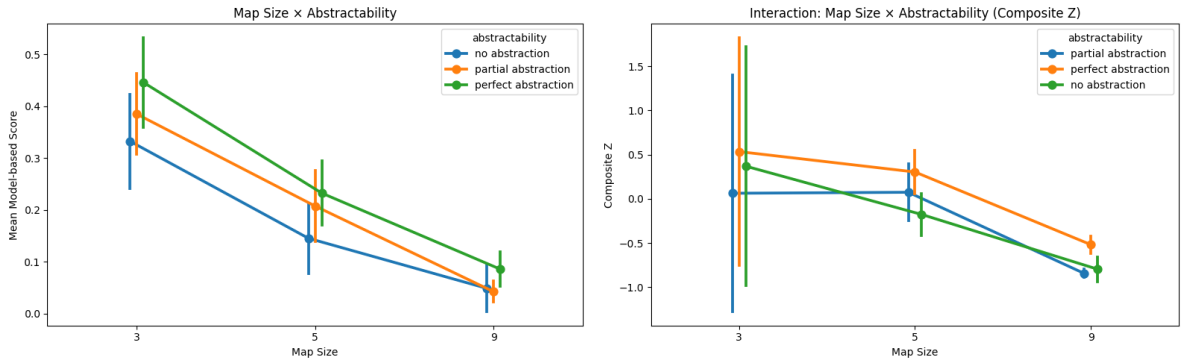
(a) Perfect abstraction **(b)** Partial abstraction **(c)** No abstraction

Figure 5.10: Maps of size 5 by 5 showing varying levels of abstractability.



(a) Perfect abstraction **(b)** Partial abstraction **(c)** No abstraction

Figure 5.11: Maps of size 9 by 9 showing varying levels of abstractability.

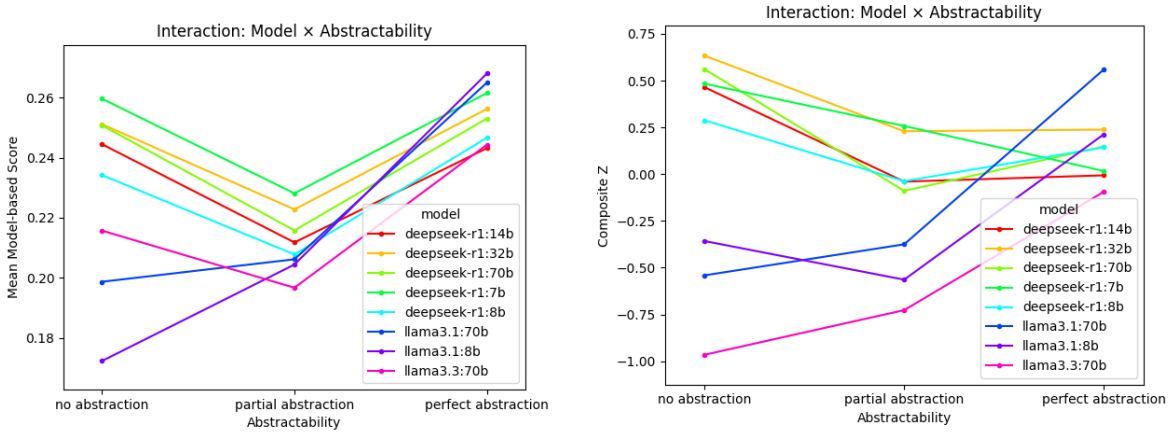


(a) Model-based metric **(b)** Composite-z metric

Figure 5.12: Interactions plots between map size and abstractability, across all models and prompts for model-based and composite-z score

Secondly, Figure 5.13 illustrates how different LLMs respond to abstractability. The model-based scores show that all models improve in performance when ideal abstractions exist, but gaps between models are more evident when no abstraction is possible. Deepseek models, maintain higher performance on average, suggesting stronger generalization capabilities or inductive priors. When ideal abstraction is possible, llama models tend to perform significantly better. This trend is even more pronounced when using the composite z-score. This once again suggests that deepseek models are better at extracting partial structure or ”filtering out” useful groupings in ambiguous environments. It re-emphasize that model architecture plays a significant role in whether useful abstractions can be recovered when the environment lacks formal symmetry.

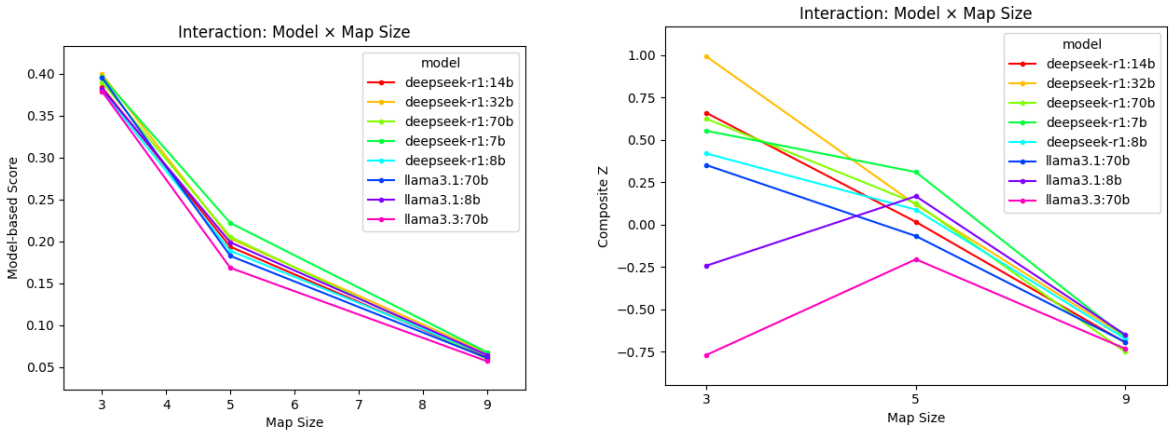
Figure 5.14 provides a breakdown of performance trends by model across different environment sizes. While all models show a decline in abstraction quality as map size increases, some models exhibit greater resilience. Notably, the deepseek models retain comparatively higher model-based and com-



(a) Model-based metric **(b)** Composite-z metric

Figure 5.13: Interaction between LLM models and abstractability level, averaged across prompts and map sizes.

posite z-scores even on the largest environments. Furthermore, the model-based metric reveals less separation between model families than the composite z-score. This again highlights that planning performance does not always align with structural similarity. Some models with modest structural scores can still support effective planning under abstraction.



(a) Model-based metric **(b)** Composite-z metric

Figure 5.14: Interactions plots between LLM models and map size, across all prompts for model-based and composite z score

5.2.4. Failures

During the experiments, several model–prompt combinations failed to produce valid abstractions. A failure was seen if a prompt could not be cleaned, and the model would have to be reprompted. In some cases, even with a full 12 hour window, some model=prompt combinations failed to generate a full set of 20 prompts and as such would need to be fully rerun.

Failures were not uniformly distributed, but concentrated around specific prompts and models. From Figure 5.15, it can be observed that prompt 10 accounted for the highest number of failures (26), followed by prompts 5 and 1. This indicates that prompt formulation plays a major role in robustness, while some prompts consistently yielded usable abstractions, others led to more frequent breakdowns.

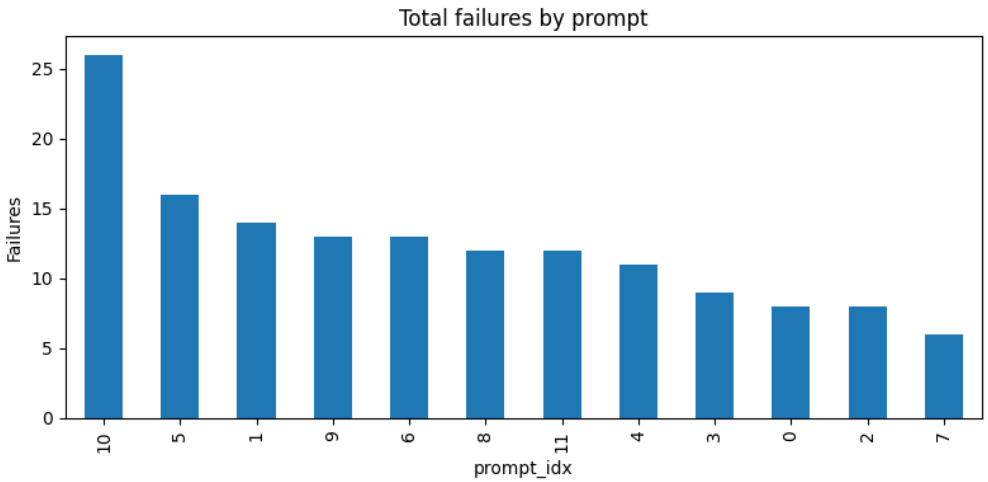


Figure 5.15: A bar graph showing the amount of times an abstraction extraction failed for each prompt

Looking at models in Figure 5.16, the Deepseek-R1:14B variant exhibited the most failures (47), followed by Deepseek-R1:70B (30) and Deepseek-R1:8B (23). While the largest models were not the most error-prone, mid-sized variants often struggled, suggesting that scaling alone does not guarantee robustness. Smaller models like LLaMA showed fewer outright failures, but as other metrics demonstrated, they also produced lower-quality abstractions overall.

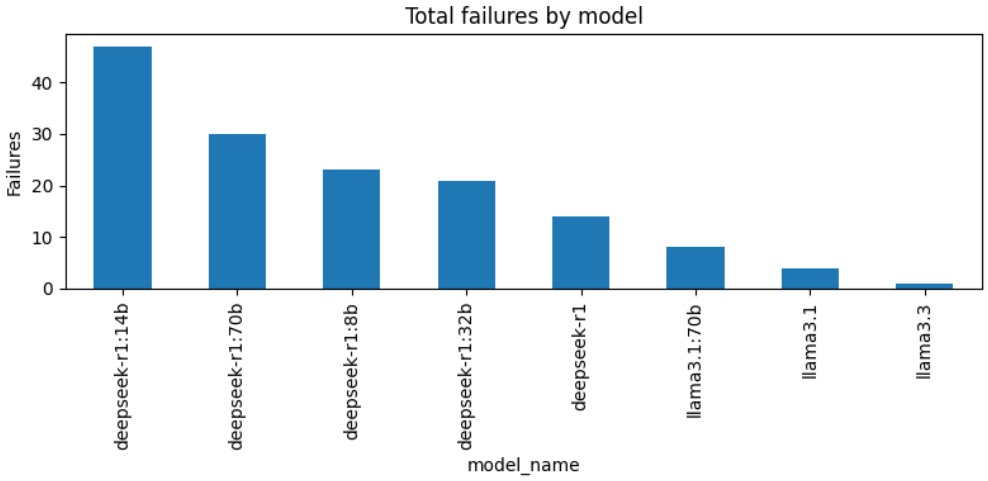


Figure 5.16: A bar graph showing the amount of times ab abstraction extraction failed for each model

The heatmap shown in Figure 5.17 shows the average runs per model–prompt combination and highlights the fact that certain model–prompt pairs were stable and converged quickly, while others required multiple runs to obtain a valid output. In some cases averages exceeding three runs. This suggests

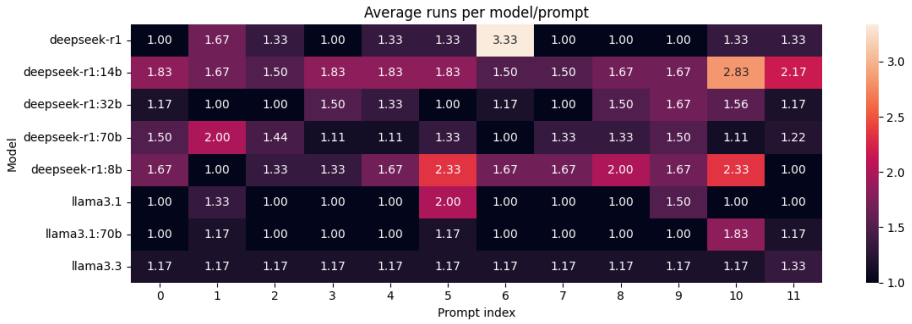


Figure 5.17: A heatmap of the average amount of runs a model-prompt combination needed before they produced 20 useable abstractions.

that certain prompt–model pairings are inherently fragile and can lead to results that were not expected, thus they could not be cleaned and evaluated in the the pipeline and were therefore discarded.

Failures also depended on environment properties. Abstractions seem to be most fragile in maps without clear abstraction structure, whereas partially or perfectly abstractable maps were slightly more robust. This indicates that environments with exploitable symmetries not only yield higher-quality abstractions but also reduce outright failures.

The failure analysis demonstrates that robustness depends on a combination of prompt design, model architecture, and environment structure. Certain prompts and mid-sized models were particularly affected, requiring multiple retries or failing entirely, while structured prompts and larger models generally performed more reliably. Moreover, environments with inherent symmetries not only supported higher abstraction quality but also reduced outright failures. These findings emphasize that both careful prompt engineering and choosing the right model–environment pairing are essential for consistent abstraction generation.

6

Conclusion

This thesis set out to explore whether LLMs can be used to generate meaningful abstractions in MDPs, where state and action spaces can grow rapidly. This work investigated whether LLMs can offer scalable and usable abstractions to simplify planning, particularly through MCTS. The approach combined prompting pipelines, structural and behavioral evaluation metrics, and agent-based rollouts across structured environments to assess the quality and utility of LLM-generated abstractions.

The central research question was decomposed into two main sub-questions to quantitatively answer the question at hand. **The results suggest that the answer to both is conditionally yes, depending on model architecture, prompt design, and environment structure.**

LLM Capabilities

The experiments confirmed that LLMs are capable of producing cluster-based abstractions that are structurally aligned with ground-truth homomorphisms in simple MDP environments. This was particularly true for the Deepseek-R1 model family, which consistently outperformed the LLaMA models across both model-based and performance-based evaluations. Notably, even smaller Deepseek models were able to match or outperform larger LLaMA variants, highlighting that model architecture and training methodology may be more important than raw parameter count. Furthermore, the replies generated by the LLMs were not directly usable in the context of the environment, which also emphasizes the need to process replies in order to adequately use LLMs for other purposes than just text-generation.

Prompt Engineering

Prompt engineering played a critical role in abstraction quality. Prompts using structured representations consistently outperformed free-text formats. Furthermore, prompts that included reasoning components in their output, asking the LLM to give a rationale why states were grouped, led to higher overall scores. Tukey's HSD analysis also confirmed that instructions derived from ChatGPT's templating offered an advantage. These findings validate and extend prior literature that emphasizes the importance of structured prompting for symbolic reasoning tasks.

Environment

As expected, the abstraction quality deteriorated with increasing environment size and decreasing abstractability. While LLMs performed reasonably well on 3×3 maps, performance collapsed on 9×9 maps, as well as maps lacking any exploitable symmetry. Despite this, Deepseek-R1 models were more resilient than LLaMA models and showed the ability to extract useful abstractions even in environments without symmetry. These results highlight the practical limits of LLM-based abstraction as well as differences between architectures. They also demonstrate that abstraction extraction under minimal supervision requires structure-aware prompt design and abstraction-aware model evaluation.

Answering the Research Questions

Can LLMs produce abstractions that are close to the optimal abstraction, if it exists?

Yes, especially when provided with structured prompts and when the environment is reasonably sized and exhibits compressible dynamics. Deepseek models can consistently approach ideal abstractions on maps with global symmetries.

Can LLMs produce abstractions that are useful for planning?

Yes, although not always. While structural similarity helps, some lower-scoring abstractions still enabled effective planning in MCTS. The composite score proved especially useful in identifying these cases. The results highlight both the promise, but also the limitations of LLM-driven abstraction. While effective in simple and structured settings, performance drops in complex ones. This suggests that LLMs may serve as useful assistants for abstraction in domains with clear, describable patterns (e.g., games, puzzles), but require augmentation, such as human guidance or fine-tuning, before being reliable in open-ended decision-making tasks.

Limitations and Future Work

This thesis focused on gridworld MDPs with deterministic transitions and a single-agent setting. Extending this work to partially observable or stochastic environments is an open challenge. Furthermore, the use of cluster-based abstractions limits the types of abstractions that can be evaluated. Future research may consider relational or option-based abstractions that are better suited for hierarchical planning.

Another limitation is scale. While the approach scaled well to 9×9 environments, abstraction quality degraded quickly beyond this point. Improvements in prompt engineering, representation learning, or few-shot prompting may alleviate this in future studies. Furthermore, the models that were investigated were limited to open-soure models and also limited in size. Further research into proprietary LLMs, larger LLMs or even fine-tuned LLMs is also still open.

Finally, while this work treated LLMs as black-box abstraction generators, incorporating more advanced prompting strategies, such as CoT reasoning chains or graph-based memory, may improve interpretability and robustness.

While still limited in scalability and reliability, the findings show that with careful prompting and structured input, LLMs can generate abstractions that are both meaningful and usable for downstream planning.

References

[1] M. Games, *4x games*, Jun. 2008. [Online]. Available: https://web.archive.org/web/20080618074603/http://www.mobygames.com/game-group/4x-games.

[2] improveTheEducationSystem, *State space complexity of chess*, Apr. 2024. [Online]. Available: https://www.chess.com/blog/improveTheEducationSystem/state-space-complexity-of-chess.

[3] Y.-J. Hsu and D. P. Liebana, "Mcts pruning in turn-based strategy games," in *AIIDE Workshops*, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:234795673.

[4] L. Li, T. J. Walsh, and M. L. Littman, "Towards a unified theory of state abstraction for mdps," in *AI&M*, 2006. [Online]. Available: https://api.semanticscholar.org/CorpusID:245037.

[5] E. van der Pol, T. Kipf, F. A. Oliehoek, and M. Welling, *Plannable approximations to mdp homomorphisms: Equivariance under actions*, 2020. arXiv: 2002.11963 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2002.11963.

[6] C. Gao, X. Lan, N. Li, *et al.*, *Large language models empowered agent-based modeling and simulation: A survey and perspectives*, 2023. arXiv: 2312.11970 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2312.11970.

[7] R. Gallotta, G. Todd, M. Zammit, *et al.*, *Large language models and games: A survey and roadmap*, 2024. arXiv: 2402.18659 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2402.18659.

[8] D. Zhang, X. Huang, D. Zhou, Y. Li, and W. Ouyang, *Accessing gpt-4 level mathematical olympiad solutions via monte carlo tree self-refine with llama-3 8b*, 2024. arXiv: 2406.07394 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2406.07394.

[9] S. Kambhampati, K. Valmeekam, L. Guan, *et al.*, *Llms can't plan, but can help planning in llm-modulo frameworks*, 2024. arXiv: 2402.01817 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2402.01817.

[10] Z. Zhao, W. S. Lee, and D. Hsu, *Large language models as commonsense knowledge for large-scale task planning*, 2023. arXiv: 2305.14078 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2305.14078.

[11] Y. Xu, W. Li, P. Vaezipoor, S. Sanner, and E. B. Khalil, *Llms and the abstraction and reasoning corpus: Successes, failures, and the importance of object-based representations*, 2024. arXiv: 2305.18354 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2305.18354.

[12] J. tse Huang, E. J. Li, M. H. Lam, *et al.*, *How far are we on the decision-making of llms? evaluating llms' gaming ability in multi-agent environments*, 2024. arXiv: 2403.11807 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2403.11807.

[13] Y. Xu, S. Wang, P. Li, *et al.*, *Exploring large language models for communication games: An empirical study on werewolf*, 2024. arXiv: 2309.04658 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2309.04658.

[14] S. Hu, T. Huang, F. Ilhan, *et al.*, *A survey on large language model-based game agents*, 2024. arXiv: 2404.02039 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2404.02039.

[15] S. Qi, S. Chen, Y. Li, *et al.*, *Civrealm: A learning and reasoning odyssey in civilization for decision-making agents*, 2024. arXiv: 2401.10568 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2401.10568.

[16] G. Konidaris, "On the necessity of abstraction," *Current Opinion in Behavioral Sciences*, vol. 29, pp. 1–7, 2019, Artificial Intelligence, ISSN: 2352-1546. DOI: https://doi.org/10.1016/j.cobeha.2018.11.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352154618302080.

[17] A. Bai, S. Srivastava, and S. Russell, “Markovian state and action abstractions for mdps via hierarchical mcts,” English (US), *IJCAI International Joint Conference on Artificial Intelligence*, vol. 2016-January, pp. 3029–3037, 2016, 25th International Joint Conference on Artificial Intelligence, IJCAI 2016 ; Conference date: 09-07-2016 Through 15-07-2016, ISSN: 1045-0823.

[18] T. Miller, *Modelling and abstraction for mdps*, [Accessed 10-01-2025], 2023. [Online]. Available: \url{https://gibberblot.github.io/rl-notes/single-agent/modelling-and-abstraction.html}.

[19] D. Han, *Projects | hdg94.github.io*, https://hdg94.github.io/projects/, [Accessed 10-01-2025], 2024.

[20] H. Wang, S. Dong, and L. Shao, “Measuring structural similarities in finite mdps,” in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, ser. IJCAI’19, Macao, China: AAAI Press, 2019, pp. 3684–3690, ISBN: 9780999241141.

[21] J. Taylor, “Lax probabilistic bisimulation,” English, Master’s Thesis, McGill University, 2008. [Online]. Available: https://escholarship.mcgill.ca/concern/theses/m613n220q.

[22] B. Ravindran and A. G. Barto, “Smdp homomorphisms: An algebraic approach to abstraction in semi-markov decision processes,” in *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, 2003, pp. 1011–1016. [Online]. Available: https://api.semanticscholar.org/CorpusID:7157818.

[23] G. Jeh and J. Widom, “Simrank: A measure of structural-context similarity,” in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’02, New York, NY, USA: Association for Computing Machinery, 2002, pp. 538–543, ISBN: 158113567X. DOI: 10.1145/775047.775126. [Online]. Available: https://doi.org/10.1145/775047.775126.

[24] Álvaro Visús, J. García, and F. Fernández, *A taxonomy of similarity metrics for markov decision processes*, 2021. arXiv: 2103.04706 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2103.04706.

[25] Henry, *A brief introduction of alphago and deep learning: How it works*, https://medium.com/@kinwo/a-brief-introduction-of-alphago-and-deep-learning-how-it-works-76e23f82fe99, [Accessed 11-01-2025], Oct. 2017.

[26] P. Rajkumar, “A survey of monte-carlo techniques in games master ’ s scholarly paper,” M.S. thesis, University of Maryland, 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:15434160.

[27] Mciura, *Steps of monte-carlo tree search*, Retrieved from Wikipedia., 2013. [Online]. Available: https://en.wikipedia.org/wiki/Monte_Carlo_tree_search.

[28] L. Xu, A. Dockhorn, and D. Perez-Liebana, “Elastic monte carlo tree search,” *IEEE Transactions on Games*, vol. 15, no. 4, pp. 527–537, 2023. DOI: 10.1109/TG.2023.3282351.

[29] L. Xu, J. Hurtado-Grueso, D. Jeurissen, D. P. Liebana, and A. Dockhorn, *Elastic monte carlo tree search with state abstraction for strategy game playing*, 2022. arXiv: 2205.15126 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2205.15126.

[30] Y. Xu, E. B. Khalil, and S. Sanner, *Graphs, constraints, and search for the abstraction and reasoning corpus*, 2022. arXiv: 2210.09880 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2210.09880.

[31] H. Sun, Y. Zhuang, L. Kong, B. Dai, and C. Zhang, *Adaplanner: Adaptive planning from feedback with language models*, 2023. arXiv: 2305.16653 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2305.16653.

[32] E. AI, *Llama 3.3 vs deepseek-r1*. [Online]. Available: https://www.edenai.co/post/llama-3-3-vs-deepseek-r1.

[33] W. Hua, L. Fan, L. Li, *et al.*, *War and peace (waragent): Large language model-based multi-agent simulation of world wars*, 2024. arXiv: 2311.17227 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2311.17227.

[34] A. Madaan, N. Tandon, P. Gupta, *et al.*, *Self-refine: Iterative refinement with self-feedback*, 2023. arXiv: 2303.17651 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2303.17651.

[35] M. AI, *Introducing meta llama 3: The most capable openly available llm to date*, https://ai.meta.com/blog/meta-llama-3/, [Accessed 11-01-2025], Apr. 2024.

[36] M. AI, *Introducing llama 3.1: Our most capable models to date*, https://ai.meta.com/blog/meta-llama-3-1/, [Accessed 11-01-2025], Jul. 2024.

[37] Ollama, *Llama3.3*, https://ollama.com/library/llama3.3, [Accessed 11-01-2025], Dec. 2024.

[38] Ollama, *Llama3.2*, https://ollama.com/library/llama3.2, [Accessed 11-01-2025], Sep. 2024.

[39] DeepSeek, *Deepseek*. [Online]. Available: https://www.deepseek.com/en.

[40] DeepSeek-AI, D. Guo, D. Yang, *et al.*, *Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning*, 2025. arXiv: 2501.12948 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2501.12948.

[41] ollama, *Deepseek-r1*. [Online]. Available: https://ollama.com/library/deepseek-r1.

[42] Q. Ye, M. Axmed, R. Pryzant, and F. Khani, *Prompt engineering a prompt engineer*, 2024. arXiv: 2311.05661 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2311.05661.

[43] D. Jeurissen, D. Perez-Liebana, J. Gow, D. Cakmak, and J. Kwan, *Playing nethack with llms: Potential & limitations as zero-shot agents*, 2024. arXiv: 2403.00690 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2403.00690.

[44] Microsoft, *Introducing nlweb: Bringing conversational interfaces directly to the web*, 2025. [Online]. Available: https://news.microsoft.com/source/features/company-news/introducing-nlweb-bringing-conversational-interfaces-directly-to-the-web/.

[45] J. Long, *Large language model guided tree-of-thought*, 2023. arXiv: 2305.08291 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2305.08291.

[46] M. Besta, N. Blach, A. Kubicek, *et al.*, "Graph of thoughts: Solving elaborate problems with large language models," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 16, pp. 17682–17690, Mar. 2024, ISSN: 2159-5399. DOI: 10.1609/aaai.v38i16.29720. [Online]. Available: http://dx.doi.org/10.1609/aaai.v38i16.29720.

[47] D. Kahneman, *Thinking fast and slow*. Penguin Books, 2011.

[48] W. News, *Microsoft introduces browser-level ai apis at build 2025: Transforming web development with in-browser ai*, 2025. [Online]. Available: https://www.windowsnews.ai/article/microsoft-introduces-browser-level-ai-apis-at-build-2025-transforming-web-development-with-in-browser-ai.366900.

[49] N. Gregoire and M. Bouillot, 1998. [Online]. Available: https://cgm.cs.mcgill.ca/~godfried/teaching/cg-projects/98/normand/main.html.

[50] [Online]. Available: https://www.rust-lang.org/.

[51] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. The MIT Press, 2018.

[52] Delft High Performance Computing Centre (DHPC), *DelftBlue Supercomputer (Phase 2)*, https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2, 2024.

[53] Avnish, *Deepseek-r1 671b: Complete hardware requirements*, 2025. [Online]. Available: https://dev.to/askyt/deepseek-r1-671b-complete-hardware-requirements-optimal-deployment-setup-2e48.

[54] W. M. T., *Gpu system requirements for running deepseek-r1*, 2025. [Online]. Available: https://apxml.com/posts/gpu-requirements-deepseek-r1.

A

Homomorphism Function (Rust)

This appendix includes the code used in the rust backend to determine the ideal abstraction for a given map. This function is called when running the agents, building matrices for scoring and for generating representations. As these calculations are computationally expensive, it has been trimmed to be as parallel as possible (only really possible with the signature function) and uses early stopping to avoid excessive computations. For the purposes of this thesis (maps up to sizes 10 by 10), this algorithm has worked very well, however certain early stopping parameters might need to be tweaked if the state space grows.

```rust
use crate::core::abstraction::*;
use crate::core::game::{utils::actions::Action, utils::errors::GameError, *};
use errors::AbstractionError;
use game_logic::Game;
use ordered_float::OrderedFloat;
use rayon::prelude::*;
use state::State;
use std::collections::{HashMap, HashSet, VecDeque};
use std::time::Instant;
use storing::{load_cache, save_cache, AbstractionEntry};

/// BFS to enumerate all reachable states from the initial game state.
/// Each unique state is assigned a unique index from [0, N].
/// N is the number of reachable states.
pub fn get_all_states(game: &Game) -> Result<Vec<State>, GameError> {
    // Hashset to track visited states
    let mut visited_states: HashSet<State> = HashSet::new();
    let mut possible_states = Vec::new();

    // Standard BFS queue starting with the root game state
    let mut state_queue = VecDeque::from([game.get_state()]);

    while let Some(current_state) = state_queue.pop_front() {
        // Skip states that have already been visited
        if visited_states.contains(&current_state) {
            continue;
        }

        // Mark state as visited
        visited_states.insert(current_state.clone());
        possible_states.push(current_state.clone());

        // Find new reachable states based on actions --> append to queue
        for action in current_state.valid_moves() {
            let new_state = match game.simulate(&current_state, &action) {
                Ok((state, _)) => state,
                Err(e) => return Err(e),
            };

            state_queue.push_back(new_state);
```

```
        }
    }

    // Assign each state an index
    for (index, state) in possible_states.iter_mut().enumerate() {
        state.index = Some(index as isize);
    }

    Ok(possible_states)
}

/// Compute the ""signature of one state under the current partitioning.
/// For each action, record (reward, next_partition_id).
/// Sorting these pairs gives us a fingerprint used to decide which states are equivalent.
/// This version is specifically so it can be used in parallel with rayon.
pub fn compute_signature_parallel(
    state: &State,
    partition: &HashMap<isize, usize>,
    game: &Game,
    position_lookup: &HashMap<(usize, usize), isize>,
) -> Vec<(OrderedFloat<f32>, usize)> {
    let mut outcomes = vec![];

    // For each action, simulate and record (reward, partition_of_successor).
    for action in state.valid_moves.iter() {
        // Simulate next game state
        // WARNING: simulated game states do not have an index therefore mapping to indexes
            is done with unit position
        // Couldn't think of a better way to do this
        let (_, vars) = game.simulate(state, action).unwrap();
        let pos = game.simulate(state, action).unwrap().0.unit_position;

        // Get next state index from mapping to unit position
        let next_index = match position_lookup.get(&pos) {
            Some(idx) => *idx,
            None => {
                eprintln!("Simulated state not found in state set.");
                eprintln!("Missing position: {:?}", pos);
                eprintln!(
                    "State details: {:?}",
                    game.simulate(state, action).unwrap().0
                );
                panic!("Abstraction failed: new state was not found in state set.");
            }
        };

        // Find which partition that successor state currently belongs to.
        let partition_id = *partition
            .get(&next_index)
            .expect("Partition must contain all state indices");

        // Push the reward + partition pair.
        outcomes.push((OrderedFloat(vars.score), partition_id));
    }

    // Sort so that signature is order-invariant across action enumeration.
    outcomes.sort_by(|a, b| a.partial_cmp(b).unwrap());
    outcomes
}

/// Main loop of -MDPhomomorphism refinement:
///   1. Initialize coarse partition: terminal vs. nonterminal.
///   2. Repeat until (no change) or early-stop:
///        a) For *each* state, compute its signature.
///        b) Group states by identical signature.
///        c) Reassign each group a new unique partition id.
///   3. Return the final clusters of -stateindices.
pub fn compute_mdp_homomorphism(states: &[State], game: &Game) -> Vec<Vec<isize>> {
    let mut partition: HashMap<isize, usize> = HashMap::new();

    // Start with two partitions: -goalstates (pid=0) vs. everything else (pid=1)
```

```
    for state in states.iter() {
        let done = game.goal() == state.unit_position;
        partition.insert(
            state.index.expect("State must be indexed"),
            if done { 0 } else { 1 },
        );
    }

    // Build unit_position -> index lookup table
    // States that come from simulation don't have an index so we need to compare unit positions since they are unique
    // Used in the signature function, this spares us time having to recalculate it
    let mut position_lookup: HashMap<(usize, usize), isize> = HashMap::new();
    for state in states.iter() {
        position_lookup.insert(state.unit_position, state.index.unwrap());
    }

    // Heuristic early stop variables - tuned by just playing around
    // IMPORTANT HERE:
    // `min_iters`: Minimum amount of iterations before we early stop
    // `max_stagnant_iters`: How many iterations we see barely any change before we stop
    let total_states = states.len();
    let mut changed = true;
    let mut iteration = 0;
    let min_iters = 10000;
    let max_stagnant_iters = 100;
    let mut stagnant_count = 0;
    let mut prev_partition_count = 0;

    // Refinement
    // O(S * A * number of iterations)
    // Computationally very expensive, therefore using early stopping
    while changed {
        // Compute signatures in parallel with rayon to distribute over cores
        let sig_state_pairs: Vec<(Vec<(OrderedFloat<f32>, usize)>, isize)> = states
            .par_iter()
            .map(|state| {
                let sig = compute_signature_parallel(state, &partition, game, &position_lookup);
                (sig, state.index.unwrap())
            })
            .collect();

        // Regroup states by identical signature
        let mut groups_by_signature: HashMap<Vec<(OrderedFloat<f32>, usize)>, Vec<isize>> = HashMap::new();
        for (sig, idx) in sig_state_pairs {
            groups_by_signature.entry(sig).or_default().push(idx);
        }

        // Build a new partition map by assigning each group a new pid
        let mut new_partition: HashMap<isize, usize> = HashMap::new();
        let mut pid = 0;

        for group in groups_by_signature.values() {
            for idx in group.iter() {
                new_partition.insert(*idx, pid);
            }
            pid += 1;
        }

        // Get the new partitions and see how they have changed compared to the last cycle
        let new_partition_count = new_partition.len();
        if iteration >= min_iters {
            if new_partition_count == prev_partition_count {
                stagnant_count += 1;
            } else {
                stagnant_count = 0;
            }

```

```
            // If we get a `somewhat` stable grouping after `n` iterations we can assume its gtg
            // If partially or fully abstractable we would be able to converge to a different grouping over time
            if stagnant_count >= max_stagnant_iters {
                println!(
                    "Early stop after {} stagnant iterations ({} total states, {} groups)",
                    stagnant_count, total_states, new_partition_count
                );
                break;
            }
        }

        prev_partition_count = new_partition_count;
        iteration += 1;

        if new_partition == partition {
            changed = false;
        } else {
            partition = new_partition;
        }
    }

    // Convert final partition map into Vec<Vec<isize>>
    let mut groups: HashMap<usize, Vec<isize>> = HashMap::new();
    for (idx, group_id) in partition {
        groups.entry(group_id).or_default().push(idx);
    }

    // Group the abstraction by ground state index
    // Similar to how it was done in Python prototype to allow comparing the results
    let mut clusters: Vec<Vec<isize>> = groups
        .into_values()
        .map(|mut v| {
            v.sort_unstable();
            v
        })
        .collect();
    clusters.sort_unstable_by_key(|cluster| cluster[0]);

    println!("Required {} iterations to converge", iteration);

    clusters
}

/// Top-level API: either load from cache to save time or compute the exact homomorphism.
/// Returns (all_states, clusters), and saves to disk for next time.
pub fn get_abstraction(game: &Game) -> Result<(Vec<State>, Vec<Vec<isize>>), AbstractionError> {
    let now = Instant::now();
    let config = &game.world_configuration();
    let cache_file = "abstraction_cache.json";

    // Load cache if available
    let mut cache = load_cache(cache_file).map_err(AbstractionError::Io)?;

    // Look for config
    if let Some(entry) = cache.iter().find(|e| &e.config == config) {
        return Ok((entry.states.clone(), entry.clusters.clone()));
    }

    // No config so we get all states and run compute function
    let mut game_clone = game.clone();
    let all_states =
        get_all_states(&mut game_clone).map_err(|e| AbstractionError::Computation {
            error: e.to_string(),
        })?;

    let clusters = compute_mdp_homomorphism(all_states.as_slice(), game);

    // Save to file
```

```
    cache.push(AbstractionEntry {
        config: config.clone(),
        states: all_states.clone(),
        clusters: clusters.clone(),
    });
    println!("Saving config...");
    save_cache(cache_file, &cache).map_err(AbstractionError::Io)?;

    let elapsed_time = now.elapsed();
    println!(
        "Took {} seconds to calculate exact homomorphism",
        elapsed_time.as_secs()
    );

    Ok((all_states, clusters))
}
```

B

Scoring Function (Python)

This appendix contains the scoring function that is outlined in **??**. It uses the bisimulation metrics and Hausdorff distance to determine how close MDPs are and returns a score in [0, 1] enabling the classification of abstractions.

```python
import numpy as np
from scipy.stats import wasserstein_distance


def bisimulation_similarity(
    candidate_clustering: list[list[int]],
    ideal_clustering:     list[list[int]],
    transitions:          np.ndarray,   # shape (S, A, S)
    rewards:              np.ndarray,   # shape (S, A)
    c:                    float = 0.5   # trade-off for Wasserstein distance
) -> float:
    """
    Compute a [0,1] similarity between two abstractions of an MDP (candidate vs ideal)
    using a -bisimulationstyle distance (García et al. 2022). 1.0 means identical,
    0.0 is as far apart as possible.

    Steps:
      1. Build the abstract MDP for each clustering:
         - r_hat[i,a]: average reward of all ground states in abstract state i under action a
         - T_hat[i,a,j]: probability of transitioning from abstract i to abstract j under
             action a

      2. For every pair of abstract states (i in candidate, j in ideal):
         - Compute the -worstcase -actionwise distance:
           dist(i,j) = max_a [ -(1c)·|r_c[i,a-]r_i[j,a]| + c·Wasserstein(T_c[i,a ,], ^T_i[j,a
               ,]) ]

      3. Lift to a -fullMDP distance via the directed Hausdorff:
         d_M = max( max_i min_j dist(i,j),  max_j min_i dist(i,j) )

      4. Map distance → similarity in [0,1] by 1/(1 + d_M).
    """

    # Precompute abstract MDP from matrices given by the rust_core library
    def build_abstract_mdp(clusters: list[list[int]]):
        """
        Internal helper function that, given a partition of S -groundstates into K clusters,
            returns:
            T_hat: (K, A, K)  aggregated transition probabilities
            r_hat: (K, A)     aggregated rewards
        """

        K = len(clusters)
        S, A, _ = transitions.shape

```

```
        # Initialize matrices
        T_hat = np.zeros((K, A, K), dtype=float) # T_hat[i,a,j] = avg_{s in Ci} sum_{s' in Cj} T[s,a,s']
        r_hat = np.zeros((K, A), dtype=float) # r_hat[i,a]    = avg_{s in Ci} R[s,a]

        # Iterate each abstract state
        for i, C in enumerate(clusters):
            Csize = max(len(C), 1)

            # sum rewards
            # rewards[C, :] has shape (|C|, A)
            r_hat[i] = rewards[C, :].sum(axis=0) / Csize

            # sum transitions
            # transitions[C, a, :] has shape (|C|, S)
            # we want for each a, the total flow into each Cj
            for a in range(A):

                P_sprime = transitions[C, a, :].sum(axis=0) / Csize # P_sprime[s'] = avg over s in C of T[s,a,s']

                # Aggregate P_sprime over each cluster Cj
                for j, Cj in enumerate(clusters):

                    T_hat[i, a, j] = P_sprime[Cj].sum() # sum P_sprime[s'] over s' in Cj

        return T_hat, r_hat

    # Build both candidate and ideal abstract MDPs
    T_cand, r_cand = build_abstract_mdp(candidate_clustering)
    T_ideal, r_ideal = build_abstract_mdp(ideal_clustering)
    Kc, A, _ = T_cand.shape
    Ki, _, _ = T_ideal.shape

    # -Abstractstate ""locations on the line for Wasserstein metric
    positions_c = np.arange(Kc)
    positions_i = np.arange(Ki)

    # Compute pairwise -abstractstate distances d_S[i,j]
    # d_S[i,j] will hold the -worstcase (over actions) distance between i and j
    d_S = np.zeros((Kc, Ki), dtype=float)

    for i in range(Kc):
        for j in range(Ki):

            # worst-case over actions
            max_over_a = 0.0
            for a in range(A):

                # reward difference
                rd = abs(r_cand[i, a] - r_ideal[j, a])

                # transition distance via 1-Wasserstein
                # https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.wasserstein_distance.html
                P1 = T_cand[i, a, :]
                P2 = T_ideal[j, a, :]
                td = wasserstein_distance(
                    positions_c, positions_i,
                    P1, P2
                )

                # weighted combination
                dist_ia = (1 - c) * rd + c * td
                if dist_ia > max_over_a:
                    max_over_a = dist_ia

            d_S[i, j] = max_over_a

    # Lift to -fullMDP distance via (directed) Hausdorff

```

```
    # for each candidate state i, find closest ideal j
    row_max = np.max(np.min(d_S, axis=1)) if Kc > 0 else 0.0

    # for each ideal state  j, find closest candidate i
    col_max = np.max(np.min(d_S, axis=0)) if Ki > 0 else 0.0

    d_M = max(row_max, col_max)

    # Map into similarity [0,1]
    similarity = 1.0 / (1.0 + d_M)  # perfect match   d_M=0   sim=1.0

    return float(similarity)
```

C

Prompt Elements (YAML)

This appendix includes all the considered prompt elements for this thesis. The prompt elements are divided into instructions, necessary context, context and output.

```yaml
# This file contains all prompt composition elements. Prompts are comprised of these elements. Each element has an id (unique) and value.
# Each prompt is comprised of:
# 1 instruction: each prompt needs an instruction i.e. call to action what to do
# 1 necessary context: a context that is needed to describe the nature of the problem. Anything marked with necessary
# n contexts: can use any amount of contexts you want to help the LLM produce a good result.
# 1 output: a proper prompt needs to specify the output that is desired for usage / futher processing

instruction:
  - id: "basic1"
    val: "Please group these states into abstract states."
  - id: "basic2"
    val: "Each state presents a unique position for the player in a grid world. Please group the states into abstract states."
  - id: "basic3"
    val: "Each state presents a unique position for the player in a grid world. Please group the states without coding."
  - id: "basic4"
    val: "Each state presents a unique position for the player in a grid world. Do not return any code."
  - id: "cot1"
    val: "Think step by step and group these states into abstract states."
  - id: "cot2"
    val: "Each state presents a unique position for the player in a grid world. Think step by step and group these states into abstract states."
  - id: "relation1"
    val: "Group these states based on their spatial relation and behavior."
  - id: "relation2"
    val: "Each state presents a unique position for the player in a grid world. Group these states based on their spatial relation and behavior."
  - id: "role1"
    val: "You are an expert in decision-making working on a grid world environment."
  - id: "role2"
    val: "You are a decision-making assistant working in a grid world environment."
  - id: "role3"
    val: "Imagine you are teaching a new student about state abstraction in a grid world."
  - id: "gpt1"
    val: "Please create an abstraction by grouping states into clusters. The abstraction should reflect symmetries in state connectivity and their relation to the goal."
  - id: "gpt2"
    val: "Consider that you have prior knowledge that corner states and edge states behave differently than center states."
  - id: "hierarchy1"
```

```
    val: "Construct a hierarchical abstraction of the 3x3 state space into at least two
        levels: a fine-grained intermediate abstraction and a more coarse final abstraction."

necessary_context:
  - id: "necessary-domain1"
    val: "The grid world always has the goal located in the bottom right corner of the map."
  - id: "necessary-domain2"
    val: "The goal is always at the last state. Movements are allowed up/down/left/right if
        not blocked by the grid boundary."
  - id: "domain2"
    val: "Some tiles may represent obstacles that the agent cannot traverse."

context:
  - id: "domain2"
    val: "Some tiles may represent obstacles that the agent cannot traverse."
  - id: "domain3"
    val: "Adjacent states in the grid are connected based on valid movements in the cardinal
        directions (up, down, left, right)."
  - id: "performance1"
    val: "Efficient state abstraction helps reduce the computational complexity during
        planning."
  - id: "performance2"
    val: "The ideal abstraction should enable the agent to solve the game."
  - id: "performance3"
    val: "Your response will be benchmarked against the pre-calculated optimal solution."
  - id: "efficiency"
    val: "The agent must navigate the grid efficiently."
  - id: "role1"
    val: "Explain your reasoning step-by-step as if instructing a learner. Make sure the
        explanation is accessible and non-technical."
  - id: "background1"
    val: "An abstract state is a group of ground states that can lead to the same
        distribution of next abstract states under symmetries of actions."
  - id: "background2"
    val: "An abstract state is a group of ground states that can lead to the same
        distribution of next ground states under symmetries of actions."
  - id: "background3"
    val: "A good abstraction considers the symmetries in the world that can be exploited to
        group ground states."
  - id: "background4"
    val: "A good abstraction considers the symmetries in both states and actions."
  - id: "test1"
    val: "Each abstract state should correspond to a set of original states that function
        similarly from the agents perspective, both structurally (adjacent connections) and
        in terms of distance to the goal."
  - id: "test2"
    val: "Using the 3x3 grid with states 0 to 8, propose an abstraction that leverages these
        known differences."
  - id: "test3"
    val: "States are considered equivalent if the optimal actions and expected value from
        following the policy are effectively the same. Based on this principle, cluster the
        states into abstract states."
  - id: "assumption"
    val: "When clustering states into abstract states, consider not only the state layout but
         also the action set (up, down, left, right). Show how certain states are
        interchangeable if we rotate or reflect the grid, keeping the action semantics
        aligned."

output:
  - id: "out1"
    val: "Do not return any text, only the grouping of the states."
  - id: "out2"
    val: "Do not return any text, only the grouping in form list[list[int]]."
  - id: "out3"
    val: "Output the groups as a list of state clusters."
  - id: "out4"
    val: "Output the rationale for the state groupings alongisde the clustering itself."
  - id: "out5"
    val: "Provide the final abstract clusters."
```

D

Full Prompt Templates

This appendix includes all the prompts, generated for a simple 3x3 map without any obstacles, to showcase how these prompts look like.

Prompt 0

```
You are an expert in decision-making working on a grid world environment.
The goal is always at the last state. Movements are allowed up/down/left/right if not blocked by the grid boundary.
Some tiles may represent obstacles that the agent cannot traverse.
0 2 5
1 4 7
3 6 8
Do not return any text, only the grouping in form list[list[int]].
```

Prompt 1

```
You are an expert in decision-making working on a grid world environment.
The goal is always at the last state. Movements are allowed up/down/left/right if not blocked by the grid boundary.
Some tiles may represent obstacles that the agent cannot traverse.
{"goal":8,"grid":[[0,2,5],[1,4,7],[3,6,8]],"start":0}
Do not return any text, only the grouping in form list[list[int]].
```

Prompt 2

```
You are an expert in decision-making working on a grid world environment.
The grid world always has the goal located in the bottom right corner of the map.
States are considered equivalent if the optimal actions and expected value from following the policy are effectively the same. Based on this principle, cluster the states into abstract states.
0 2 5
1 4 7
3 6 8
Do not return any text, only the grouping of the states.
```

Prompt 3

```
You are an expert in decision-making working on a grid world environment.
The grid world always has the goal located in the bottom right corner of the map.
States are considered equivalent if the optimal actions and expected value from following the policy are effectively the same. Based on this principle, cluster the states into abstract states.
{"goal":8,"grid":[[0,2,5],[1,4,7],[3,6,8]],"start":0}
Do not return any text, only the grouping of the states.
```

Prompt 4

```
Group these states based on their spatial relation and behavior.
The goal is always at the last state. Movements are allowed up/down/left/right if not blocked by the grid boundary.
States are considered equivalent if the optimal actions and expected value from following the policy are effectively the same. Based on this principle, cluster the states into abstract states.
0 2 5
1 4 7
3 6 8
Do not return any text, only the grouping in form list[list[int]].
```

Prompt 5

```
Group these states based on their spatial relation and behavior.
The goal is always at the last state. Movements are allowed up/down/left/right if not blocked by the grid boundary.
States are considered equivalent if the optimal actions and expected value from following the policy are effectively the same. Based on this principle, cluster the states into abstract states.
{"goal":8,"grid":[[0,2,5],[1,4,7],[3,6,8]],"start":0}
Do not return any text, only the grouping in form list[list[int]].
```

Prompt 6

```
Each state presents a unique position for the player in a grid world. Group these states based on their spatial relation and behavior.
The goal is always at the last state. Movements are allowed up/down/left/right if not blocked by the grid boundary.
Your response will be benchmarked against the pre-calculated optimal solution.
0 2 5
1 4 7
3 6 8
Do not return any text, only the grouping in form list[list[int]].
```

Prompt 7

```
Each state presents a unique position for the player in a grid world. Group these states based on their spatial relation and behavior.
The goal is always at the last state. Movements are allowed up/down/left/right if not blocked by the grid boundary.
Your response will be benchmarked against the pre-calculated optimal solution.
{"goal":8,"grid":[[0,2,5],[1,4,7],[3,6,8]],"start":0}
Do not return any text, only the grouping in form list[list[int]].
```

Prompt 8

```
Imagine you are teaching a new student about state abstraction in a grid world.
The grid world always has the goal located in the bottom right corner of the map.
Adjacent states in the grid are connected based on valid movements in the cardinal directions (up, down, left, right).
0 2 5
1 4 7
3 6 8
Do not return any text, only the grouping in form list[list[int]].
```

Prompt 9

```
Imagine you are teaching a new student about state abstraction in a grid world.
The grid world always has the goal located in the bottom right corner of the map.
Adjacent states in the grid are connected based on valid movements in the cardinal directions (up, down, left, right).
{"goal":8,"grid":[[0,2,5],[1,4,7],[3,6,8]],"start":0}
Do not return any text, only the grouping in form list[list[int]].
```

Prompt 10

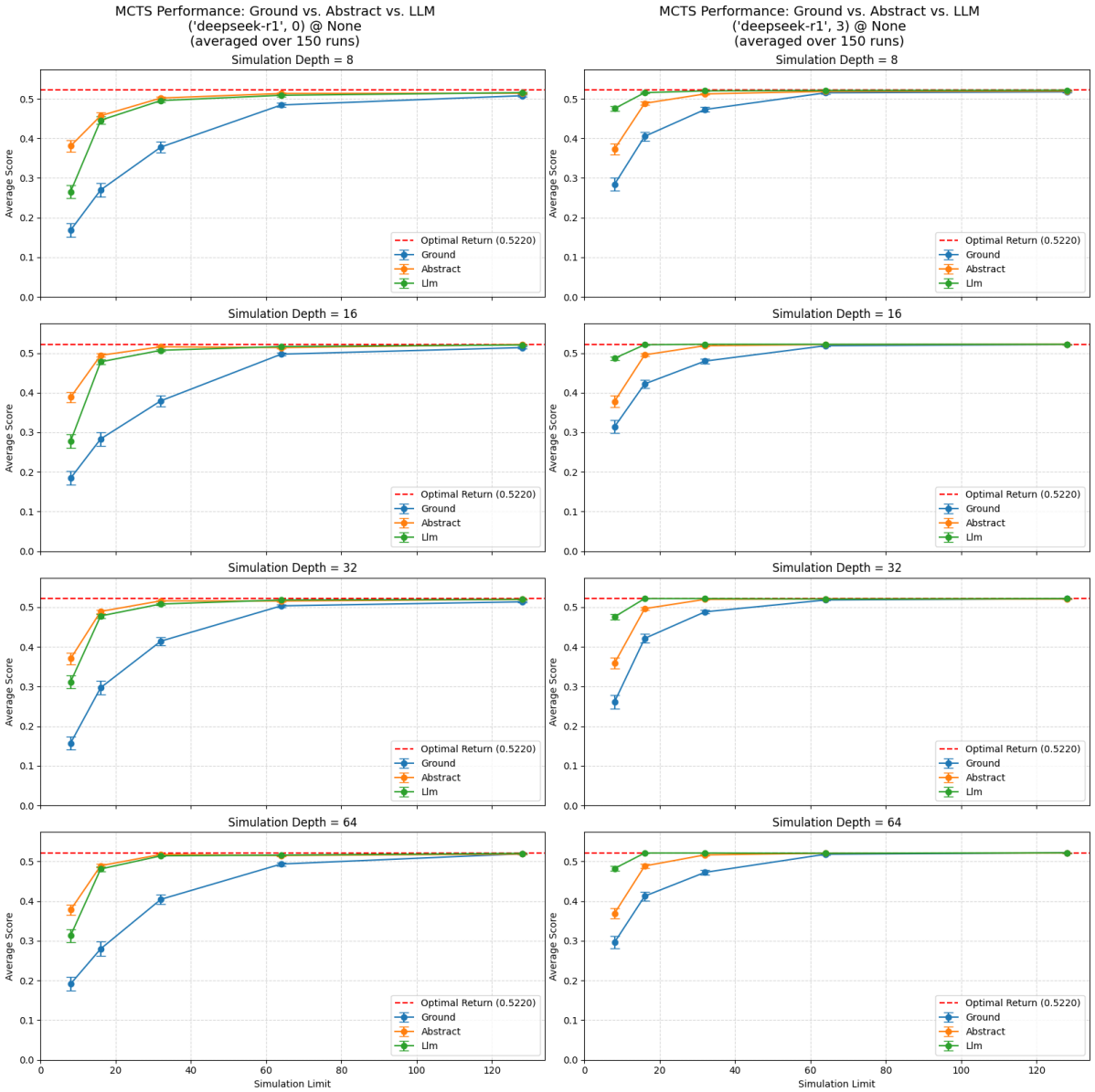
```
Please create an abstraction by grouping states into clusters. The abstraction should reflect symmetries in state connectivity and their relation to the goal.
Some tiles may represent obstacles that the agent cannot traverse.
Each abstract state should correspond to a set of original states that function similarly from the agents perspective, both structurally (adjacent connections) and in terms of distance to the goal.
States are considered equivalent if the optimal actions and expected value from following the policy are effectively the same. Based on this principle, cluster the states into abstract states.
When clustering states into abstract states, consider not only the state layout but also the action set (up, down, left, right). Show how certain states are interchangeable if we rotate or reflect the grid, keeping the action semantics aligned.
0 2 5
1 4 7
3 6 8
Output the rationale for the state groupings alongisde the clustering itself.
```

Prompt 11

```
Please create an abstraction by grouping states into clusters. The abstraction should reflect symmetries in state connectivity and their relation to the goal.
Some tiles may represent obstacles that the agent cannot traverse.
Each abstract state should correspond to a set of original states that function similarly from the agents perspective, both structurally (adjacent connections) and in terms of distance to the goal.
States are considered equivalent if the optimal actions and expected value from following the policy are effectively the same. Based on this principle, cluster the states into abstract states.
When clustering states into abstract states, consider not only the state layout but also the action set (up, down, left, right). Show how certain states are interchangeable if we rotate or reflect the grid, keeping the action semantics aligned.
{"goal":8,"grid":[[0,2,5],[1,4,7],[3,6,8]],"start":0}
Output the rationale for the state groupings alongisde the clustering itself.
```

E

MCTS Performances



(a) MCTS results of deepseek-r1:7b with prompt 0 on a partially abstractable map

(b) MCTS results of deepseek-r1:7b with prompt 3 on a perfectly abstractable map

Figure E.1: MCTS results of deepseek-r1:7b using different prompts on different maps