SAARLAND UNIVERSITY

Faculty of Mathematics and Computer Science
Department of Computer Science
MASTER THESIS

Leveraging LLMs for Guiding Exploration in Reinforcement Learning

Submitted by
Vaibhav Jain
Saarbrücken
December 2025

Advisor: Dr. Gerrit Großmann
Reviewer 1: Prof. Verena Wolf
Reviewer 2: Prof. Sebastian Vollmer

Advisor:
Dr. Gerrit Großmann
Data Science and its Applications (DSA) Lab
German Research Center for Artificial Intelligence
Kaiserslautern, Germany

Reviewer 1:
Prof. Verena Wolf
Modelling and Simulation Lab
Saarland University
Saarbrücken, Germany

Reviewer 2:
Prof. Sebastian Vollmer
Data Science and its Applications (DSA) Lab
German Research Center for Artificial Intelligence
Kaiserslautern, Germany

Saarland University
Faculty MI – Mathematics and Computer Science
Department of Computer Science
Campus - Building E1.1
66123 Saarbrücken
Germany

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne die Beteiligung dritter Personen verfasst habe, und dass ich keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.
Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus Veröffentlichungen oder aus anderweitigen fremden Äußerungen entnommen wurden, sind als solche kenntlich gemacht.
Ich erkläre mich damit einverstanden, dass die Arbeit mittels eines Plagiatsprogrammes überprüft wird.
Mir ist bewusst, dass der Verstoß gegen diese Versicherung zum Nichtbestehen der Prüfung bis hin zum Verlust des Prüfungsanspruchs führen kann.

Declaration of Original Authorship

I hereby declare that I have written this thesis independently and without the involvement of third parties, and that I have used no sources or aids other than those indicated. All passages taken directly or indirectly from publications or other external sources have been identified as such.
I agree that the thesis may be checked using plagiarism detection software.
I am aware that any violation of this declaration may result in failing the examination and lead to losing the right to be examined.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik an der Universität des Saarlandes aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department at Saarland University.

Ort/Place, Datum/Date	Unterschrift/Signature

Acknowledgements

No one climbs Everest alone. Likewise, this thesis would not have been possible without the support of many people.

First, I would like to thank my advisor, Dr. Gerrit Grossmann, for his constant guidance, valuable input, and genuine interest, which shaped the direction of this research.

Next, I would like to express my gratitude to my friends here in Saarbrücken. I am fortunate to have met so many wonderful people here that I cannot possibly name them all. Still, I would like to mention a couple of people: Shreyas, who has been through thick and thin with me for almost ten years now and was also mentioned in my bachelor's thesis acknowledgment, and Eva, who has changed my life in the most wonderful ways and has been a constant source of encouragement.

I would also like to thank my parents, Sanjay and Neelam, for their unconditional love and for providing me with every opportunity in life, and my sister, Vaishali, for her unwavering support.

Finally, I am grateful to everyone who, in ways big and small, contributed to this work and to my growth.

Abstract

This thesis investigates how LLMs can guide exploration in RL without imposing hard constraints on decision-making. We hypothesize that LLMs encode broad world knowledge and commonsense procedural regularities that, when elicited via prompts, can bias exploration while preserving policy autonomy. We propose a soft-constraint integration where LLM-generated suggestions are added as structured *hints* within the agent's observation, with a hint-availability flag. This design preserves the original MDP, leaves the learning objective unchanged, and lets the policy learn when to use or ignore guidance.

Methodologically, we develop a prompting and encoding pipeline that translates compact state summaries into schema-constrained outputs suitable for consumption by standard RL policies. The approach is algorithm-agnostic; we instantiate it with PPO and evaluate across domains with varying structure and difficulty: Minigrid, TicTacToe, and Deal or No Deal. Supplementary evaluations with DQN and REINFORCE are also provided.

Empirically, hints elicited via structure-preserving prompts (and chain-of-thought where appropriate) are reliable and context-relevant. In Minigrid, integrating hints as soft inputs improves sample efficiency and, on harder tasks, final performance over tabula-rasa baselines while remaining below an oracle upper bound. In compact domains such as TicTacToe and Deal or No Deal, prompts yield interpretable, valid suggestions (e.g., improved action validity under masking and approximately 85% agreement with curated data in Deal or No Deal), with training gains bounded by short horizons and small state spaces. Across settings, policies learn to discount misleading hints, demonstrating robustness to imperfect guidance.

We discuss limitations, primarily the computational overhead of frequent LLM queries, and outline cost-aware extensions including adaptive hint scheduling, distillation, and lightweight serving. Overall, the results support LLM-guided hints as a practical, robust mechanism to accelerate learning in sufficiently complex RL tasks while maintaining policy autonomy.

Keywords: Reinforcement Learning, Large Language Models, Exploration, Prompt Engineering, Policy Robustness, Sample Efficiency

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Motivation

Reinforcement Learning (RL) provides a general framework for sequential decision making in which an agent learns to act by maximizing expected cumulative reward through interaction with an environment [51]. In standard setups, the agent starts with little (or none) *a priori* knowledge beyond the environment interface and must discover effective behaviours from scratch. This *tabula-rasa* setup is general, but it is often sample-inefficient, especially in sparse-reward and long-horizon settings where successful trajectories are rare and hard to uncover through undirected exploration.

Often, exploration breaks down when rewards are sparse, action spaces are large or combinatorial, or environment dynamics are complex. In open-world environments like Minecraft and household simulators such as VirtualHome (and Crafter), a purely exploratory policy may waste time on illogical actions, for example, attempting `Eat Cow` instead of `Drink Cow` or `Put Coffee in Microwave` before picking it up, which yields little learning signal (Fig. 1.1) [19, 14, 46].

Humans, in contrast, approach new tasks with rich priors (linguistic, perceptual, and conceptual) that guide early hypotheses and structure exploration. Recent advances in LLMs suggest a complementary path. Trained on broad textual corpora, these models encode procedural regularities and causal patterns that can support planning and commonsense reasoning [8]. The central idea in this thesis is to use this implicit knowledge to guide RL agents by augmenting the information available for decision making.

1.2 Previous Approaches

Prior work has investigated several routes for combining LLMs with RL. One line of work uses LLMs as reward models to evaluate trajectories or shape rewards via preference learning [11, 28]. These methods can provide dense evaluative signals in sparse-reward

Figure 1.1: Environments that benefit from commonsense-guided exploration: Crafter (left) and VirtualHome (right).

settings but risk mis-specification and reward hacking [4] when model judgements deviate from task goals.

Another direction employs LLMs as hierarchical planners that generate high-level sub-goals or action sketches to be executed by lower-level controllers [3]. This can substantially structure exploration but often creates a tight dependency: if sub-goals are inaccurate or inconsistent, the downstream policy may be steered off-course with limited capacity to override the plan.

A third approach uses LLMs as direct action policies or action assistants [32], proposing primitive actions at each step. While this can inject strong priors quickly, it typically imposes a hard constraint on decision making: the RL agent must closely follow or heavily weight the LLM's choices during training.

Across these approaches, a common limitation emerges. They often create *hard constraints* that require LLM guidance to be consistently accurate. When the LLM's understanding is limited or its suggestions are wrong, tight coupling can hurt performance because the agent has little autonomy to down-weight or ignore poor advice.

1.3 Research Objectives

To address the limitations of hard-constraint approaches, this thesis investigates a *soft-constraint* alternative. We hypothesise that LLM guidance can be integrated as *structured hints* supplied via the observation channel rather than as directives or reward modifications. Consider a simple grid-world task where rewards are only provided upon success. A purely exploratory agent may require many episodes to stumble upon a successful sequence. At the same time, providing hints such as "turn right" or "move forward", derived from an LLM's analysis of the encoded state, can bias exploration toward plausible behaviours in early training.

Specifically, the agent's observation is extended with structured hints and a hint-availability

indicator. Let o_t denote the environment observation at time t, h_t the LLM-provided hint (e.g. a primitive action token encoded for the policy), and $h_t^{\text{avail}} \in \{0, 1\}$ a flag indicating whether a hint is present. We define the enhanced observation

$$o'_t = \{\, o_t,\ h_t,\ h_t^{\text{avail}} \,\}. \tag{1.1}$$

Hints are not necessarily free-form natural language; they are provided as structured inputs compatible with the agent's observation space. When no hint is supplied, h_t assumes a neutral value and $h_t^{\text{avail}} = 0$, preserving a consistent input structure. This design is algorithm-agnostic and can be used with standard RL methods without modifying learning objectives.

This framing leads to two main research questions:

RQ1: Prompt-Engineered Game Knowledge in LLMs How can prompt engineering encode task-specific dynamics, rules, and constraints into an LLM so that, given compact state summaries, it reliably generates structured, state-contingent action hints suitable for the agent's observation space?

RQ2: LLM-Hints for RL Training To what extent do LLM-generated hints, integrated as soft inputs in an augmented observation space (Eq. (1.1)), improve RL training in terms of sample efficiency, final performance, and robustness to imperfect or absent guidance compared to tabula-rasa baselines?

1.4 Contributions

This thesis makes the following contributions:

- A formalisation and implementation of LLM-derived *hints* as enhanced observations for RL, including a hint-availability mechanism that enables soft, learnable reliance on guidance.
- A practical prompting and encoding pipeline that integrates task knowledge into LLMs and outputs structured hints suitable for policy consumption.
- An empirical analysis of robustness to imperfect guidance, highlighting when and how agents learn to ignore misleading hints while benefiting from useful ones.
- Systematic evaluation across tasks with varying difficulty, analysing final performance and sample efficiency, and comparing different hint schedules and representations.

By shifting from hard constraints to enhanced observations, the proposed design aims to retain the flexibility of standard RL while leveraging LLM planning capabilities to improve learning efficiency.

1.4.1 Scope and Positioning

Scope. This thesis studies LLM-provided guidance as *optional, bounded-size* inputs to the observation channel without altering rewards, transition dynamics, or the learning objective. Hints are structured encodings (e.g., a one-hot action token or a categorical

distribution over actions) accompanied by an availability flag, enabling the policy to learn when to rely on or ignore them. The design is algorithm-agnostic; we instantiate it with standard RL algorithms (e.g., PPO, DQN, REINFORCE) while preserving the original task specification.

Assumptions and non-goals. We assume access to an LLM that can be prompted from compact state summaries; we do not assume perfect, consistent, or unbiased guidance. We do not perform reward modelling, preference learning, or trajectory cloning, and we do not require test-time dependence on the LLM once a policy is trained, unless explicitly evaluated.

Positioning. Our approach differs from imitation learning (which optimises to match expert actions), curriculum learning (which reshapes task difficulty), and reward shaping (which modifies the objective). Instead, we preserve the original MDP and provide auxiliary context that the agent can exploit or ignore, yielding a soft-constraint integration that maintains autonomy and robustness to imperfect guidance.

1.5 Thesis Structure

The remainder of this thesis is organised as follows. Chapter 2 provides an overview of the technical background relevant to the algorithms and integration strategies studied in this work. Chapter 3 surveys prior work on LLM-RL integration and heuristic guidance methods. Chapter 4 details the technical foundations and system architecture for injecting LLM-hints into RL pipelines. Chapter 5 presents empirical results. The thesis concludes in Chapter 6 by summarising findings, discussing limitations, and suggesting directions for future research.

Chapter 2
Technical Background

This chapter provides the technical foundations used throughout the thesis. It reviews RL concepts relevant to the learning setup, summarizes training techniques that address variance and stability, and introduces LLMs with emphasis on transformer architectures and prompting. The background supports the design choices and algorithms described later and applies to both policy- and value-based RL methods.

2.1 Reinforcement Learning Background

2.1.1 Problem Setup and Notation

We model sequential decision making using a Markov decision process (MDP) defined by the tuple (S, A, T, R, γ). Here S is the set of states, A is the set of actions, $T(s' \mid s, a)$ is the transition function describing environment dynamics, $R(s, a)$ is the reward function, and $\gamma \in (0, 1]$ is the discount factor that downweights future returns [51]. At each time t, the agent observes a state $s_t \in S$ (or an observation o_t in partially observed settings), selects an action $a_t \in A$ according to its policy $\pi(a \mid s)$, and receives a reward $r_t = R(s_t, a_t)$, after which the environment transitions to a new state $s_{t+1} \sim T(\cdot \mid s_t, a_t)$.

The agent's objective is to maximize the expected discounted return, defined as the random variable $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ and the performance measure

$$J(\theta) \;=\; \mathbb{E}_{\pi_\theta}\big[G_0\big], \tag{2.1}$$

where π_θ is a parameterized policy with parameters θ. Value functions assess the long-term utility of states and actions: the state-value function $V^\pi(s) = \mathbb{E}_\pi[G_t \mid s_t{=}s]$ and the action-value function $Q^\pi(s, a) \;=\; \mathbb{E}_\pi[G_t \;\mid\; s_t{=}s, a_t{=}a]$. The advantage function $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$ quantifies the relative benefit of action a compared to the baseline value at s.

2.1.2 Core Challenges

The following recurring challenges motivate the design choices and techniques used in this thesis:

- **Sparse or delayed rewards**: Long horizons and infrequent feedback make credit assignment difficult and slow down learning.
- **Exploration inefficiency**: In large or combinatorial state/action spaces, undirected (e.g., ϵ-greedy) exploration rarely finds informative experience. This is especially true for tasks requiring semantic understanding or high-level planning to discover sparse rewards, motivating the need for external guidance.
- **Stability with function approximation**: Bootstrapping and non-stationary targets can introduce divergence or collapse without appropriate stabilization.

These challenges underlie the variance-reduction, stabilization, and exploration strategies summarized in the next subsection.

2.1.3 RL Training Techniques

Effective training combines estimators and stabilizers to balance bias, variance, and sample efficiency:

Monte Carlo vs Temporal-Difference

Monte Carlo (MC) methods use complete returns to provide unbiased but high-variance targets, while temporal-difference (TD) methods bootstrap from learned estimates to reduce variance at the cost of bias. Many practical algorithms interpolate between MC and TD to manage this trade-off.

On-Policy vs Off-Policy

On-policy methods update using data generated by the current policy, simplifying objectives but limiting sample reuse. Off-policy methods learn from replayed or behavior-policy data, improving data efficiency while requiring importance weighting or architectural safeguards to control distribution shift.

Variance Reduction

Control variates such as baselines reduce estimator variance without biasing gradients. Advantage estimation focuses learning on action advantages rather than raw returns. Generalized Advantage Estimation (GAE) exponentially weights multi-step TD errors to trade bias for variance and improve credit assignment.

Regularization and Stabilization

Entropy bonuses encourage exploration and prevent premature collapse; reward and observation normalization improve numerical conditioning; gradient clipping bounds update magnitudes. Additional techniques include standardized advantages and clipped objectives that limit policy drift between updates.

Practicalities

Target networks provide slowly moving bootstrap targets; replay buffers decorrelate samples in value-based methods; observation scaling and running statistics improve stationarity. These choices interact with algorithm design and are tuned to the environment's dynamics and reward scale.

2.1.4 Policy-Based Methods

Policy-based, also called policy gradient, methods search the space of parametric stochastic policies $\pi_\theta(a \mid s)$ by performing gradient ascent on the expected return $J(\theta) = \mathbb{E}_{\pi_\theta}[G_0]$. Under the regularity conditions in [51], the *policy gradient theorem* yields the exact expression

$$\nabla_\theta J(\theta) \;=\; \mathbb{E}_{\pi_\theta}\big[\, \nabla_\theta \log \pi_\theta(a \mid s)\; Q^\pi(s,a) \,\big], \tag{2.2}$$

where the expectation is taken over the on-policy visitation distribution and Q^π is the action-value function. Key consequences:

- **Unbiasedness with function approximation.** Eq. (2.2) holds for any differentiable policy representation (e.g., deep networks), avoiding the divergence issues that plague value-gradient methods when the function class cannot represent the true value function.
- **High variance.** Monte-Carlo estimates of Q^π introduce large variance that grows with episode length. *Baselines* $b(s)$ can be subtracted without bias ([51, Sec. 13.2]), leading to the advantage form $A^\pi(s,a) = Q^\pi(s,a) - b(s)$ and significant variance reduction.
- **Credit assignment.** The log-derivative term $\nabla_\theta \log \pi_\theta(a \mid s)$ attributes credit to parameters influencing the probability of chosen actions. When combined with temporal-difference critics (actor–critic), gradients become semi-bootstrapped and less noisy.

In practice, trajectories are collected with the current policy, advantages are estimated using either Monte-Carlo returns or Generalised Advantage Estimation (GAE), and parameters are updated by stochastic gradient ascent (or a trust-region variant). Classical algorithms include:

- **REINFORCE**: episodic Monte-Carlo gradient with a configurable baseline.
- **Actor–Critic (A2C/A3C)**, a learned state-value critic reduces variance and enables asynchronous data collection.
- **PPO / TRPO**: constrained updates via clipping or KL penalties to prevent destructive large steps.

The next subsections detail these representatives and their specific optimization choices.

REINFORCE

REINFORCE is an episodic Monte-Carlo policy-gradient method. For a trajectory $\tau = (s_0, a_0, r_0, \ldots, s_T)$, define the reward-to-go $\hat{G}_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_k$. Using a (state-dependent)

baseline $b(s_t)$ to reduce variance without bias, the gradient estimator is

$$\nabla_\theta J(\theta) \;=\; \mathbb{E}_{\pi_\theta}\Big[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\,\big(\hat{G}_t - b(s_t)\big)\Big]. \tag{2.3}$$

Equivalently, one minimizes the negative surrogate

$$L^{\text{REINFORCE}}(\theta) \;=\; \mathbb{E}_t\big[-\log \pi_\theta(a_t \mid s_t)\,\hat{A}_t\big], \tag{2.4}$$

where $\hat{A}_t = \hat{G}_t - b(s_t)$; common choices include $b(s_t) = V_\phi(s_t)$ or a moving-average baseline. Entropy regularization is often added to encourage exploration. REINFORCE is simple and unbiased but can suffer from high variance and slow convergence on long-horizon tasks.

Actor–Critic (A2C/A3C)

Actor–critic methods learn a parametric critic to approximate the value function and use it as a baseline for lower-variance policy-gradient updates. Synchronous (A2C) and asynchronous (A3C) variants differ in how trajectories are collected and aggregated for updates.

Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) addresses instability by limiting the magnitude of policy updates through a clipped objective. Let $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ denote the probability ratio under the updated and old policies, and let $\hat{A}_t$ be an advantage estimate. The clipped surrogate objective is

$$L^{\text{CLIP}}(\theta) \;=\; \mathbb{E}_t\Big[\min\big(r_t(\theta)\,\hat{A}_t,\; \text{clip}\big(r_t(\theta), 1{-}\epsilon, 1{+}\epsilon\big)\,\hat{A}_t\big)\Big], \tag{2.5}$$

where ϵ controls the allowable deviation from the old policy. PPO combines this surrogate with value-function regression and entropy regularization, yielding a stable, sample-efficient algorithm widely used in practice. The framework in Chapter 4 augments observations with LLM hints and is fully compatible with policy-based methods. The optimization objective and update logic are unchanged, while the policy and value networks learn how to use the additional input.

2.1.5 Value-Based Methods

Q-Learning (Tabular Recap)

Q-learning iteratively applies the Bellman optimality update to estimate $Q^*(s, a)$ using transitions (s, a, r, s'). The update targets $r + \gamma \max_{a'} Q(s', a')$, driving values toward the optimal fixed point under suitable exploration and learning rates.

Deep Q-Networks (DQN)

Deep Q-Networks approximate the optimal action-value function $Q^*(s, a)$ with a neural network $Q(s, a; \theta)$. Training minimizes the temporal-difference loss with a fixed target

network and experience replay to stabilize learning:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\Big[\big(r + \gamma \max_{a'} Q_{\text{target}}(s', a'; \theta^-) - Q(s, a; \theta)\big)^2\Big]. \tag{2.6}$$

Key stabilizers include a periodically updated target network θ^- and a replay buffer to decorrelate samples. Common extensions such as Double DQN, Dueling Networks, and Prioritized Replay further improve stability and data efficiency. Much like its policy-gradient counterparts, this value-based approach is fully compatible with the proposed framework. The Q-network would simply learn to estimate $Q(o'_t, a)$, taking the augmented observation $\{o_t,\ h_t,\ h_t^{\text{avail}}\}$ as its state input.

2.2 Large Language Models Background

2.2.1 Transformer Architectures

Transformers [57] model sequences using blocks composed of self-attention and position-wise feed-forward networks, connected by residual pathways and layer normalization. The core of the architecture is the self-attention mechanism, which allows a token to weigh the importance of all other tokens in its context and exchange information.

This mechanism operates on three vectors derived from each token's input embedding: the Query (Q), the Key (K), and the Value (V). The Query vector represents the token's current information need; the Key vector represents what information a token has; and the Value vector represents the information that will be aggregated.

For a set of queries Q, keys K, and values V, the scaled dot-product attention is

$$\text{Attention}(Q, K, V) \;=\; \text{softmax}\!\left(\frac{QK^\top}{\sqrt{d_k}}\right) V, \tag{2.7}$$

where d_k is the key dimension used to stabilize gradients by scaling the dot products. Concretely, each query computes similarity scores with all keys; these scores are normalized with a softmax to obtain weights, which then form a weighted sum of the values. Multi-head self-attention repeats this operation in parallel with different learned projections, enabling the model to jointly attend to information from multiple representation subspaces.

Positional encodings (e.g., sinusoidal or rotary) inject order information otherwise absent from the permutation-invariant attention mechanism. While encoder–decoder variants support sequence-to-sequence tasks, decoder-only models are commonly used for text generation. These use masked self-attention that prevents tokens from attending to future positions, making them suitable for next-token prediction and, in turn, for generative prompting.

2.2.2 Training LLMs

Pretraining optimizes next-token prediction over large text corpora with subword tokenization (e.g., byte-pair encoding). Scale (parameters, data, compute) strongly influences performance; optimization typically uses Adam-family optimizers with learning-rate schedules and regularization. After pretraining, instruction tuning adapts models to follow natural-language instructions using supervised fine-tuning. Lightweight adapters

(e.g., LoRA) can reduce compute and memory during adaptation. Alignment methods such as preference optimization aim to shape model behavior toward human-desired responses.

2.2.3 Inference and Serving

At inference, decoding strategies trade diversity for determinism: greedy sampling maximizes likelihood at each step; temperature and top-k/top-p sampling introduce stochasticity; beam search explores multiple high-likelihood continuations. Practical serving considerations include context length limits, key–value caching to reuse attention states across steps, and batching to improve throughput, all of which affect latency and cost.

2.2.4 Prompting Techniques

Prompting is the primary method for interacting with and steering the behavior of pretrained LLMs without updating their parameters. By carefully designing the input text (the "prompt"), a user can condition the model to perform specific tasks and expose latent knowledge.

Simple prompts may include direct instructions (zero-shot) or a few examples of the task (few-shot). More advanced techniques aim to improve reasoning quality; Chain-of-Thought (CoT) prompting [60] encourages stepwise intermediate reasoning by asking the model to "think step-by-step," and self-consistency aggregates multiple reasoning chains to improve robustness.

For integration with computational systems, the prompt's output format is as important as its content. Constrained or structured prompting instructs the model to return responses in a precise, machine-readable format (e.g., fixed vocabularies or JSON schemas). This is essential for action-oriented use with an RL agent, which expects guidance as a discrete action from a predefined set or a compact numeric vector rather than ambiguous natural language. In this thesis, structured prompting is the mechanism that translates an agent's observation o_t into the actionable, structured hint h_t used in later chapters.

2.3 Relevance to This Thesis

2.3.1 Observation Augmentation with LLM-Derived Hints

At configurable decision points, the environment observation is encoded for the LLM, which returns a structured suggestion ("hint"). The agent's input is augmented as $o'_t = \{o_t,\ h_t,\ h_t^{\text{avail}}\}$, where $h_t^{\text{avail}} \in \{0, 1\}$ indicates whether a valid hint is present. This design exposes guidance as information and allows the policy to learn when to use or ignore it based on returns.

2.3.2 Compatibility with Policy- and Value-Based Algorithms

Because hints enter through the observation, the learning objective and update rules remain unchanged. Policy-gradient methods consume o'_t in place of o_t when computing

action probabilities and advantages; value-based methods estimate $Q(o'_t, a)$ similarly. No reward shaping or planner imposition is required, preserving autonomy and robustness.

2.3.3 Practical Considerations

Query frequency controls the rate of LLM interaction and can be tuned to balance guidance coverage against computational cost. Latency is mitigated through batching and key engineering choices in state encoding. The availability flag h_t^{avail} ensures consistent network inputs even when no hint is requested or returned.

In summary, this chapter covered the foundational concepts of reinforcement learning and large language models, highlighted key training techniques and transformer-based architectures, and introduced the approach of augmenting observations with LLM-derived hints in an algorithm-agnostic way. The following chapter will place these techniques in context by reviewing related work on RL, LLMs, and their integration.

Chapter 3
Related Work

This chapter reviews work at the intersection of RL and LLMs. It first outlines the fundamental RL challenges that motivate using external knowledge sources. It then traces how LLMs evolved into powerful reasoning engines, while noting their inherent limitations. The core of the chapter is a systematic survey and critique of existing LLM–RL integration paradigms. It closes by synthesizing these strands to identify a key research gap and positions the proposed "LLM-hints" framework as a grounded solution.

3.1 Foundational Challenges in RL

RL provides a powerful and general framework for sequential decision-making, enabling agents to learn optimal behaviors through trial-and-error interaction with an environment [51, 33]. Combined with deep neural networks, Deep Reinforcement Learning (DRL) has achieved remarkable successes, from mastering complex games like Go to controlling robotic systems [51]. Yet the practical application of DRL to many real-world problems remains hindered by several persistent challenges. In particular, exploration in sparse-reward environments, poor sample efficiency, and a lack of robust generalization underscore the need for methods that inject more structured information and prior knowledge into the learning process [64, 5].

3.1.1 Exploration in Sparse-Reward Settings

A primary obstacle for many RL algorithms is learning in environments with sparse rewards. In such settings, an agent receives a non-zero reward only upon reaching a specific, often distant, goal state or completing a complex sequence of actions. Most interactions yield uninformative, zero-valued feedback [2, 29, 54]. This sparsity creates a formidable exploration problem, as the probability of discovering a rewarding trajectory through random or naive strategies is combinatorially small [2]. The agent is effectively searching for a needle in a haystack, with no signal to guide its search.

This challenge is compounded by the **credit assignment problem**, the difficulty of determining which specific actions in a long sequence were critical for achieving a delayed reward [52]. Without intermediate feedback, it is nearly impossible to distinguish between purposeful and accidental success, making it hard to reinforce the correct behaviors [2]. Sparsity also exacerbates the classic **exploration-exploitation dilemma**. With little to no positive feedback to exploit, the agent is forced into a prolonged and often inefficient exploration phase, potentially never discovering states that lead to a reward [52].

To mitigate the challenges of sparse rewards, researchers have developed several classes of techniques, each with its own set of trade-offs.

- **Intrinsic Motivation:** These methods create dense, internal reward signals to encourage specific exploratory behaviors. For instance, curiosity-driven approaches reward the agent for visiting novel states or for making actions that lead to surprising outcomes, encouraging more thorough exploration [52, 45]. While these strategies can accelerate learning, the generated intrinsic rewards are often generic and may not be optimally aligned with the extrinsic task goal, leading the agent to explore irrelevant parts of the state space.
- **Reward Shaping:** This technique manually engineers a dense, auxiliary reward function to provide more frequent feedback. A common example is a reward based on the negative Euclidean distance to a target location [54, 41]. While effective, designing a good shaping reward requires significant domain expertise, is highly problem-specific, and must be done carefully to avoid altering the optimal policy of the original task [41]. Poorly designed shaping rewards can lead to "reward hacking," where the agent maximizes the auxiliary reward without solving the intended problem, potentially leading to unintended behaviors [2].
- **Hierarchical Reinforcement Learning (HRL):** HRL tackles long-horizon sparse-reward problems by decomposing a complex task into a hierarchy of more manageable, shorter-horizon subtasks [51, 52, 44]. A high-level policy learns to select a sequence of subgoals, while one or more low-level policies learn to achieve them. This temporal abstraction can make credit assignment and exploration more tractable [44]. However, HRL often relies on predefined or manually specified subgoals, and the automatic discovery of useful subtask hierarchies remains a significant open problem [51].
- **Hindsight Experience Replay (HER):** HER reframes learning by treating failed trajectories as successful attempts at achieving a different, unintended goal. After an episode where the agent fails to reach the intended goal g, HER stores the trajectory in the replay buffer not only with goal g but also with the final state s_T that was actually reached. By relabeling the goal, it derives a dense learning signal from an otherwise unsuccessful episode, dramatically improving sample efficiency in multi-goal settings [54].

3.1.2 Sample Inefficiency in DRL

A second, equally important challenge is sample inefficiency in DRL. DRL algorithms, especially model-free variants, often require millions or even tens of millions of interactions with the environment to learn a competent policy [64, 65]. This appetite for data often makes them impractical for real-world applications, such as robotics or industrial

control, where each interaction can be costly, time-consuming, or dangerous [51, 5, 45]. By contrast, humans can often learn new tasks with very few examples, highlighting a wide gap in learning efficiency [25].

This inefficiency is exacerbated when agents must learn directly from high-dimensional observation spaces, such as raw pixels. In such cases, the agent must simultaneously learn a meaningful state representation and a control policy using only a scarce RL signal, a task that requires an enormous amount of data [64]. On-policy versus off-policy distinctions also matter. On-policy methods, such as Proximal Policy Optimization (PPO), update the policy using only data collected from the most recent version of that policy, discarding past experience. This makes them inherently less sample-efficient than off-policy methods, like Soft Actor-Critic (SAC), which can reuse data from a large replay buffer collected over many past policies [64, 47].

Several strategies have been proposed to improve the sample efficiency of DRL.

- **Model-Based RL:** Instead of learning a policy directly, model-based methods first learn a model of the environment's dynamics ($T(s_{t+1}|s_t, a_t)$) and reward function ($R(s_t, a_t)$). This learned model can then be used to generate simulated trajectories for policy optimization, significantly reducing the need for real-world interaction [64, 65]. However, the performance of these methods is heavily dependent on the accuracy of the learned model; even small inaccuracies can compound over long planning horizons, leading to suboptimal policies.
- **Auxiliary Losses:** To aid representation learning from high-dimensional inputs, auxiliary tasks with unsupervised objectives can be added to the training process. For example, an agent might be jointly trained to reconstruct its input observation (using an autoencoder) while also optimizing the RL objective. This provides a dense, supervised signal that helps the network learn salient features, making the sparse RL signal more effective [64].
- **Transfer Learning and Meta-Learning:** These approaches aim to leverage knowledge gained from a distribution of related tasks to accelerate learning on a new task [52]. By learning a shared representation or a learning algorithm that adapts quickly, these methods can improve sample efficiency in novel environments.

3.1.3 Generalization and Robustness in DRL

Beyond exploration and efficiency, a key objective of modern RL research is to develop agents that generalize their learned skills. A common failure mode is overfitting to the specifics of the training environment. Agents may exhibit high performance during training but fail catastrophically when deployed in a slightly different, unseen scenario [5, 23]. Achieving robust generalization remains an open and active area of research.

Furthermore, deploying RL agents in the real world introduces critical concerns regarding safety and interpretability. An agent must be able to explore without causing damage to itself or its surroundings, requiring mechanisms for safe exploration [51, 5]. For high-stakes applications like medical diagnosis or autonomous driving, the ability to interpret and understand an agent's decision-making process is not just desirable but essential for trust and accountability [5].

The foundational challenges of RL—sparse rewards, sample inefficiency, and poor generalization—are not disparate issues. They are deeply interconnected symptoms of a more fundamental limitation: the **information bottleneck** inherent in learning from a

simple, scalar reward signal. A single number, delivered infrequently, is an extremely low-bandwidth channel for communicating the complex, causal structure of a task. Sparsity is an extreme case of this bottleneck, where the information channel is almost always silent. The resulting sample inefficiency is a direct consequence; because the learning signal is so impoverished, the agent must aggregate a massive volume of experience to extract a statistically meaningful gradient for policy improvement. This, in turn, leads to poor generalization, as the agent is more likely to learn spurious correlations specific to its limited successful experiences rather than the underlying principles of the task.

From this perspective, the array of solutions developed over the years—from intrinsic motivation and reward shaping to hierarchical learning—can be seen as attempts to widen this information bottleneck by injecting additional, more structured information into the learning process. This reframes the motivation for integrating external knowledge sources, such as LLMs, not merely as an incremental enhancement but as a potentially transformative method for expanding the information channel available to the RL agent.

3.2 LLMs as Reasoning and Knowledge Engines

As the limitations of learning from sparse scalar rewards became increasingly apparent, NLP was undergoing a revolution. The development and scaling of LLMs produced systems with remarkable capabilities, seemingly transcending their origins as simple text predictors to become powerful engines for knowledge retrieval and complex reasoning. This section traces the evolution of these models, examines their emergent capabilities, and critically assesses their acknowledged limitations, positioning them as a potent—but imperfect—source of guidance for RL agents.

3.2.1 From Statistical Models to Transformers

The capabilities of modern LLMs are the culmination of a multi-decade journey in computational linguistics, marked by several key paradigm shifts [57, 68].

- The **Symbolic Era (1950s–1980s)** featured systems based on complex sets of hand-written rules and conceptual ontologies [68, 20]. Systems like SHRDLU, which operated in a restricted "blocks world," and ELIZA, a chatbot that simulated a psychotherapist, demonstrated the potential for natural language interaction but were inherently brittle, inflexible, and unable to scale to open-domain language [68, 62, 61].
- The **Statistical Era (1990s–2000s)** marked a shift in NLP, moving away from hand-coded rules toward data-driven machine learning algorithms [68, 36]. Enabled by increasing computational power and larger text corpora, techniques like n-gram models, which predict the next word based on the probability distribution of the preceding $n-1$ words, became dominant [57]. This approach was more robust and scalable than symbolic methods but lacked a deeper understanding of semantics or long-range context.
- The **Neural Era (2010s)** began when deep learning architectures that had proven successful in computer vision were adapted for language tasks [57, 68]. This period saw the rise of word embeddings like Word2Vec and GloVe, which represented words as dense vectors in a continuous space, capturing semantic relationships

(e.g., "king" - "man" + "woman" ≈ "queen") [36, 38]. Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) networks, became standard for handling sequential data like text [57].

- The **Transformer Revolution (2017–Present)** was sparked by the seminal paper "Attention Is All You Need," which introduced the Transformer architecture [57]. By replacing sequential recurrence with a parallelizable self-attention mechanism, the Transformer enabled models to process entire sequences at once and more effectively capture long-range dependencies in text [36, 38]. This innovation unlocked unprecedented scalability, paving the way for pre-trained models like BERT (Bidirectional Encoder Representations from Transformers) and the GPT (Generative Pre-trained Transformer) series, trained on web-scale datasets with hundreds of billions or even trillions of parameters [57, 38].

3.2.2 Emergent Capabilities: ICL and CoT

The massive scaling enabled by the Transformer architecture has led to the discovery of **emergent abilities**, capabilities not present in smaller models but appearing spontaneously in large ones [59]. These abilities suggest a qualitative shift from simple pattern matching to more sophisticated forms of reasoning and task adaptation.

- **In-Context Learning (ICL):** LLMs like GPT-3 demonstrated the ability to perform new, unseen tasks simply by being provided with a few examples (or "shots") within the prompt itself, without any updates to the model's weights [38]. This allows for rapid, on-the-fly adaptation to a vast range of tasks.
- **Chain-of-Thought (CoT) Reasoning:** Prompting an LLM to "think step-by-step" before providing a final answer dramatically improves its performance on tasks requiring multi-step logical, mathematical, or commonsense reasoning [26, 60]. By externalizing their "reasoning" process, LLMs can break down complex problems into more manageable intermediate steps, mirroring a form of human-like cognition [21].
- **Vast World Knowledge:** By virtue of being pre-trained on a significant fraction of the public internet, LLMs have absorbed a broad, if sometimes inconsistent, repository of factual and commonsense world knowledge [45, 59, 43]. This allows them to answer questions, explain concepts, and generate plans related to a wide array of real-world activities.

3.2.3 Limitations in Planning and Verification

Despite these capabilities, it is important to recognize that LLMs are not reasoning agents in the formal sense. They are fundamentally auto-regressive models trained to predict the next token in a sequence based on statistical patterns in their training data [26, 55, 22]. This foundation gives rise to a set of well-documented limitations that make them unreliable for autonomous planning and decision-making.

- **Lack of Grounding and Causal Understanding:** An LLM's "understanding" is based on statistical correlations in text, not on a grounded, causal model of the world [55, 7]. It can generate text that sounds logical because logical text was prevalent in its training data, but it does not "understand" the principles of logic or causality itself [22, 7].

- **Hallucinations and Factual Unreliability:** LLMs are prone to "hallucinating," generating confident, plausible-sounding, but false or fabricated information [18, 67]. This is a critical failure mode for tasks that require factual accuracy or strict adherence to constraints, as seen in instances of LLMs inventing legal precedents or company policies [18, 67].
- **Failure in Autonomous Planning:** On formal planning benchmarks, LLMs perform poorly in autonomous mode. Multiple studies show that even state-of-the-art models like GPT-4 generate correct, executable, and goal-reaching plans in only a small fraction of cases, with one study reporting an average success rate of just 12% [22, 56]. Performance is also brittle; it degrades significantly if object and action names are obfuscated, a change that does not affect symbolic planners. This suggests that LLMs often rely on retrieving and adapting memorized plan fragments rather than engaging in first-principles planning [22, 56].
- **Inability to Self-Verify:** A crucial aspect of robust reasoning is verifying a solution's correctness. LLMs have proven poor at this task. Prompting an LLM to critique or refine its own plan often fails to yield improvements and can even worsen performance, as the model may incorrectly "correct" a valid solution or fail to recognize its own errors [22]. This inability to self-verify prevents reliable self-improvement loops where an LLM could generate plans, test them, and learn from mistakes.
- **Statelessness and Long-Horizon Coherence:** As stateless models, LLMs process each input independently and lack an inherent memory of past interactions. Maintaining a consistent plan or strategy over long horizons is therefore difficult, a key requirement for complex, multi-step tasks [7, 39].

The capabilities and limitations of LLMs are two sides of the same coin, both stemming from their training objective of next-token prediction on a massive text corpus. This process forces the model to internalize statistical patterns of human language, including grammar, facts, and common reasoning structures like chain-of-thought, giving it high *competence* in generating text that appears well-reasoned and plan-like. However, the same objective does not require the model to build a formal, verifiable model of logic, state dynamics, or causality. Consequently, there is no internal mechanism to ensure that generated output is factually correct, logically sound, or physically executable, leading to poor *performance* when tasked with autonomous planning.

This fundamental gap between competence and performance motivates hybrid LLM-RL systems. Such systems aim for a symbiotic relationship: the RL agent, through direct interaction with the environment, provides the grounding, verification, and feedback that the LLM lacks. In return, the LLM provides high-level semantic knowledge, commonsense priors, and strategic guidance that the RL agent struggles to discover from a sparse reward signal alone.

3.3 Survey of LLM-RL Integration

Recognition of the complementary strengths and weaknesses of LLMs and RL has spurred a rapidly evolving body of research focused on their integration. Multiple paradigms have emerged, each proposing a different way to structure the collaboration between the language-based knowledge of the LLM and the experience-based learning of

the RL agent. These approaches can be systematically categorized by the functional role the LLM plays within the traditional agent-environment loop: as a source of evaluation (Reward Model), a generator of experience (World Model/Simulator), a high-level commander (Hierarchical Planner), or, as proposed in this thesis, a collaborative advisor (Policy Prior).

Table 3.1: A Comparative Taxonomy of LLM-RL Integration Paradigms

Paradigm	Nature of Guidance	Core Mechanism	Key Strengths	Major Limitations
LLM as Reward Model [42, 6]	Evaluative (indirect)	LLM scores states/trajectories to yield a scalar reward.	Aligns with human preferences; handles fuzzy objectives.	Reward hacking; noisy/misaligned signal; expensive.
LLM as World Model / Simulator [43, 13]	Generative (indirect)	Generates rollouts or predicts next states for planning.	Higher sample efficiency; enables look-ahead.	Model bias; compounding errors; weak on stochastic dynamics.
LLM as Hierarchical Planner [3, 50]	Hard constraint (direct)	Decomposes goals into subgoals executed by a low-level policy.	Strong for long horizons; interpretable.	Brittle to wrong subgoals; inflexible.
LLM as Policy Prior (LLM-Hints) [16]	Soft constraint (direct)	Adds action suggestions to the observation; policy learns when to trust.	RL robustness; can ignore bad advice; adaptive.	Larger observations; LLM latency/cost.

3.3.1 LLMs as External Oracles

One major class of approaches uses the LLM as an external, non-intervening component that operates at arm's length from the agent's policy. In these frameworks, the LLM either serves as a judge of the agent's behavior by providing a reward signal or as a storyteller by generating a simulated world for the agent to learn in.

Under the **LLM as Reward Model** paradigm, the LLM functions as a proxy for the task objective. The agent performs actions in the environment, and the resulting trajectories or states are fed to an LLM, which is prompted to output a score reflecting their quality or desirability. This score is then used as the reward to train the RL agent [28, 11, 34]. This is the foundational mechanism behind Reinforcement Learning from Human Feedback (RLHF), where a reward model is first trained on a dataset of human preferences (e.g., humans ranking different model responses) and then used to fine-tune the LLM policy [6, 10]. Reinforcement Learning from AI Feedback (RLAIF) extends this by using a powerful "teacher" LLM to generate the preference labels, automating the process [6, 30]. The primary strength of this approach is its ability to align agents with complex, qualitative, and nuanced objectives, such as being "helpful and harmless," that are difficult to encode in a traditional, hand-crafted reward function [10, 30, 42]. However, this method is indirect and suffers from several limitations. The agent can learn to "hack" the reward model, finding policies that maximize the predicted score without achieving the intended goal. Furthermore, the process is computationally expensive, as it requires training a separate, large reward model and then performing RL on top of it [42].

In the **LLM as World Model / Simulator** paradigm, the LLM's generative capabilities are leveraged to create a simulated environment for the RL agent. The LLM is trained to predict future states or generate entire "imaginary rollouts" of state-action-reward

sequences [43, 48]. The RL agent can then be trained partially or entirely within this LLM-generated "dream" world, which can improve sample efficiency by reducing the need for costly or dangerous real-world interactions [59, 15, 13]. This approach is promising, but its primary challenge is the "reality gap." If the LLM's simulation of world dynamics is inaccurate, the policy learned within it will fail to transfer to the real environment [37]. A significant technical hurdle is bridging the semantic gap between the text-based nature of LLMs and the often numerical or physical nature of RL environments, a process that requires sophisticated grounding techniques [43].

3.3.2 LLMs as Hierarchical Planners

Perhaps the most intuitive and widely explored integration paradigm casts the **LLM as a high-level hierarchical planner**. This approach leverages the LLM's proficiency in reasoning and task decomposition to break down a complex, long-horizon goal into a sequence of simpler, more manageable subgoals. A low-level RL policy is then trained to execute this sequence [44, 3, 50].

Typically, the LLM is provided with a high-level instruction (e.g., "make coffee"). Often using chain-of-thought prompting, it generates a discrete plan, such as a list of subgoals: (1) "navigate to the kitchen," (2) "pick up the mug," (3) "operate the coffee machine" [50, 17, 66]. A low-level, goal-conditioned RL agent then learns a policy to achieve each subgoal in turn. This effectively transforms a sparse-reward problem into a series of dense-reward problems, as the agent receives a reward upon completing each intermediate step [51, 40, 58]. More advanced frameworks incorporate dynamic re-planning, where the agent can query the LLM again if it encounters an unexpected obstacle or fails to complete a subgoal, allowing a degree of error correction and adaptability [50, 58].

This hierarchical approach is effective for long-horizon tasks and enhances interpretability, as low-level actions can be understood in the context of the high-level plan [51]. However, its fundamental weakness is a **rigid, top-down dependency**. The success of the entire system is contingent on the correctness and feasibility of the plan generated by the LLM. As noted in Section 2.2.3, LLMs are inherently unreliable planners, prone to generating plans that are incomplete, illogical, or physically impossible [22, 56]. In this rigid hierarchy, a single flawed subgoal can cause the mission to fail, as the low-level agent has no mechanism to question, override, or adapt beyond the instructions it is given. This brittleness is a major obstacle to deploying such systems in complex, unpredictable environments.

3.3.3 Comparative Analysis

A synthesis of these dominant paradigms reveals a common underlying assumption: the LLM's output is treated as a form of ground truth. In reward modeling, the LLM's score *is* the objective function. In hierarchical planning, the LLM's plan *is* the set of commands to be executed. This framing overlooks the LLM's inherent fallibility. It establishes a relationship that is either evaluative (a judge) or dictatorial (a commander), but not truly collaborative.

The progression of these strategies is analogous to the evolution of human-computer interaction. Early systems relied on rigid, explicit commands, much like the hierarchical planner. Later systems incorporated user feedback to refine behavior, akin to the reward modeling paradigm. However, a more mature and effective model of collaboration, for both humans and AI, involves a peer-to-peer or advisory relationship, where one

party can offer suggestions and the other can use its own judgment and experience to decide whether to accept that advice. This flexible, collaborative model is largely absent from mainstream LLM-RL literature, highlighting a critical and underexplored research gap. There is a clear need for a framework that allows an RL agent to benefit from an LLM's vast knowledge while retaining the autonomy to learn from direct experience and, crucially, to learn when the LLM's advice is wrong.

3.4 From Human Guidance to AI Advice

The idea of an RL agent learning from external guidance is not new. A rich body of work in Interactive Reinforcement Learning (IRL) has explored how to accelerate learning by incorporating advice, historically from a human expert. This literature provides the theoretical foundation for a more collaborative LLM-RL framework, establishing formalisms for integrating external knowledge not as a rigid command, but as a "soft constraint" or a learnable policy prior. By reframing the LLM as an automated, high-knowledge (though imperfect) advisor, it is possible to draw directly on these established principles.

3.4.1 Advice-Taking Theory in RL

The subfield of advice-taking in RL was developed to address the same sample inefficiency problem that motivates LLM-RL integration. The core idea is that leveraging external expertise can significantly prune the search space and guide exploration more effectively than trial-and-error alone [35, 27, 53].

In **Interactive Reinforcement Learning (IRL)**, a human is placed "in the loop" during training to provide feedback [63, 1]. This feedback can take various forms, such as demonstrations of correct behavior, pairwise preferences between trajectories, scalar rewards, or, most relevantly, direct advice in the form of natural language instructions [53, 63, 24]. Early work showed that allowing an external observer to provide simple imperative instructions (e.g., "go left at the junction") and directly incorporating this advice into the agent's utility function can yield statistically significant gains in performance and learning speed [35, 27]. These methods suggest that even intermittent and simple forms of advice can substantially improve an agent's ability to solve complex tasks.

3.4.2 Policy Shaping

While early advice-taking methods were often heuristic, the concept of **Policy Shaping** provides a more formal framework for integrating external guidance [12]. This paradigm distinguishes itself from the more common Reward Shaping by altering the agent's policy directly, rather than indirectly through the reward signal.

- **Reward Shaping** modifies the environment's reward function, r, to a new function $r' = r + F(s, a, s')$, where F is a shaping potential. The agent's learning algorithm remains unchanged, but it now optimizes for a different objective. This is an *indirect* influence on the policy.
- **Policy Shaping**, in contrast, leaves the reward function and the underlying RL algorithm intact. Instead, it directly influences the action-selection process. The final policy, π_{final}, is formed by combining the agent's learned policy, π_{RL}, with an

advisory policy, π_{Advice}, derived from the external feedback. A common method of combination is a product, $\pi_{\text{final}} \propto \pi_{\text{RL}} \times \pi_{\text{Advice}}$ [12]. This is a *direct* influence on the policy.

The **Advise** algorithm is a notable Bayesian implementation of policy shaping [12]. It treats binary human feedback (e.g., "right" or "wrong" for a given action) as direct probabilistic labels on the optimality of actions. By maintaining a belief over the human's intended policy, it constructs an advisory policy that is robust to infrequent and even inconsistent feedback. The significance of policy shaping is that it provides a formal mechanism for treating external guidance as a **soft constraint**. The advisory policy acts as a prior, biasing action selection toward the suggested actions, but the agent is still free to choose other actions based on its own learned experience (represented by π_{RL}). The relative influence of advice versus experience is naturally balanced within the probabilistic framework.

3.4.3 From Human-in-the-Loop RL to LLM-in-the-Loop Systems

The principles and mechanisms developed for human-in-the-loop (HITL) systems provide a practical blueprint for the next generation of LLM-in-the-loop systems. The challenges are similar: the external source of knowledge (human or LLM) is insightful but also fallible, inconsistent, and sometimes unavailable. The goal is to extract maximum benefit from this guidance while retaining robustness through direct, grounded experience.

The analogy can be mapped directly:

- The **Human Advisor** is replaced by the **Pre-trained LLM**.
- **Natural Language Advice** is replaced by **LLM-generated text**, such as action suggestions or strategic hints.
- **Inconsistent, Noisy, or Biased Human Feedback** is analogous to **LLM Hallucinations, Planning Errors, and Inherent Biases**.

This mapping suggests that the rigid, hierarchical LLM-as-planner paradigm is a regression to earlier, less robust forms of interaction. A more advanced approach treats the LLM not as an infallible commander but as a knowledgeable, albeit imperfect, collaborator. The "LLM-hints" framework proposed in this thesis is a direct instantiation of this idea. It is, in effect, the first concrete implementation of the Policy Shaping paradigm where the source of the advisory policy is a large language model.

By providing LLM suggestions as part of the agent's observation space, the agent's policy network itself learns to perform the "shaping." It implicitly learns a complex, state-dependent function that weighs the LLM's advice against its own internal value estimates. When the LLM's hints have historically led to positive outcomes in similar states, the policy will learn to follow them. When the hints have been misleading, the policy will learn to ignore them. This approach connects two previously disparate lines of research: it takes the theoretical framework of Policy Shaping from the HITL-RL literature and scales it by using a modern, general-purpose, and infinitely patient advice-giver, the LLM. This provides a strong theoretical justification for the proposed method, positioning it not as a mere engineering trick, but as a principled advancement in integrating external knowledge into reinforcement learning.

3.5 Research Gap: Collaborative, LLM-Guided Exploration

The preceding review establishes three critical points: (1) foundational RL algorithms struggle in information-poor environments, creating a need for external guidance; (2) LLMs have emerged as powerful sources of world knowledge and reasoning but are unreliable as autonomous planners; and (3) dominant LLM-RL integration paradigms enforce a rigid, hierarchical relationship that is brittle to the LLM's failures. Together, these findings point to a clear research gap: the absence of a framework that enables a flexible, collaborative partnership between an LLM and an RL agent, allowing the agent to leverage LLM guidance as a soft, learnable constraint.

3.5.1 SOTA: Limits of Hierarchical and Reward-Based Methods

The current state of the art in LLM-RL integration, while promising, is defined by a fundamental trade-off. **Hierarchical planning** approaches [3, 50, 9] inject high-level strategy into an agent's behavior, effectively tackling long-horizon tasks. However, reliance on the LLM for a correct and executable plan makes them brittle; the entire system can fail if the LLM makes a single critical error [22, 56]. **Reward modeling** approaches [42, 6] offer more flexibility by providing an evaluative signal rather than a direct command, but the guidance is indirect. The agent must still discover effective policies through exploration and is susceptible to reward hacking, where it exploits flaws in the LLM-based judge rather than solving the task. Finally, **world model** approaches [43, 13] are powerful but risk learning in a "fantasy" world that does not correspond to reality, leading to policies that fail to transfer.

The core unaddressed problem across these paradigms is the assumption of LLM infallibility. They are designed as master-slave or judge-competitor relationships, not as partnerships between two distinct, imperfect intelligences. There is no established mechanism for an RL agent to leverage the LLM's prior knowledge while retaining the autonomy to validate, question, and override that knowledge based on grounded, empirical feedback from the environment.

3.5.2 Novelty of "LLM-Hints"

This thesis proposes to fill this gap with a framework centered on "LLM-hints." The core innovation is to re-conceptualize the role of the LLM's output, moving it from a command or a reward to a piece of advice.

In this framework, at each timestep, the LLM is prompted with the current state and overall goal, and it generates a suggestion for the next action. This action suggestion is not executed directly. Instead, it is encoded and concatenated with the agent's regular state observation, forming an **augmented observation space** [16]. The RL agent's policy, $\pi(a|s, \text{hint})$, then takes this augmented observation as input and outputs an action.

This mechanism transforms the LLM's guidance into a **soft constraint**. The policy network, being a universal function approximator, is free to learn any mapping from the augmented state to an action. Through standard RL training, the agent learns to correlate following the hint with future returns. If the LLM's advice is consistently helpful, the policy network attends to the hint portion of its input and produces the suggested action. If the advice is noisy, irrelevant, or detrimental in certain states, the policy learns to ignore that part of its input and rely on its own value estimates. This allows the agent to

learn *when* and *where* to trust its external advisor, creating a robust, collaborative system that combines the LLM's broad knowledge with the RL agent's grounded experience. As argued in Section 2.4, this is a practical and scalable implementation of Policy Shaping, where the "shaping" is learned implicitly by the policy network.

3.5.3 Research Questions and Hypotheses

This thesis examines the "LLM-hints" framework through two research questions that connect LLM prompt engineering with RL improvement:

RQ1: Prompt-Engineered Game Knowledge in LLMs How can prompt engineering encode task-specific dynamics, rules, and constraints so that, given compact state summaries, an LLM reliably generates structured, state-contingent action hints suitable for the agent's observation space?

RQ2: LLM-Hints for RL Training To what extent do LLM-generated hints, integrated as soft inputs in an augmented observation space (Eq. (1.1)), improve RL training in sample efficiency, final performance, and robustness to imperfect or absent guidance compared to tabula-rasa and hierarchical baselines?

Hypotheses.

- **Hypothesis 1:** Carefully designed prompts that encode task rules, action schemas, and compact state summaries can elicit structured, state-contingent hints from the LLM that are consistently task-relevant and suitable for direct inclusion in the agent's observation space.
- **Hypothesis 2:** Integrating LLM-generated hints as soft inputs (Eq. (1.1)) can improve sample efficiency and final performance over tabula-rasa and hierarchical baselines, while remaining robust by learning to down-weight or ignore low-quality or absent guidance.

By addressing these questions, the study aims to establish a principled, collaborative paradigm that leverages LLM reasoning as soft, learnable guidance within standard RL training.

Chapter 4
Methodology

4.1 Overview

This chapter introduces a methodology for integrating LLM guidance into RL via enhanced observations and soft constraints. The approach targets the core difficulty of exploration in sparse-reward, partially observed tasks by supplying periodic, structured hints derived from an LLM, while preserving the agent's autonomy and the original reward design.

The system forms a closed loop among the environment, an RL agent, and an LLM service. At regular intervals during training, the current environment observation is transformed into a compact textual representation and combined with a short action history and a task description (goal). The LLM is queried with this prompt to produce a hint aligned with the agent's discrete action space (e.g., a primitive action or subgoal). The hint is injected into the agent's input as part of an augmented, fixed-shape observation. Crucially, the agent remains the sole decision-maker. It learns from reward feedback to exploit or ignore hints depending on their estimated utility.

We implement this interaction as an algorithm-agnostic augmentation that can be used with policy-gradient or value-based methods. Specifically, the agent observes an enhanced input $o'_t = \{o_t, h_t, h_{avail,t}\}$ (Eq. (4.1)), where $h_{avail,t}$ indicates whether a valid hint was obtained at time t. When no hint is available, a neutral placeholder is supplied for h_t. Because hints enter only as input features, the optimization objective and reward function remain unchanged. The mechanism acts as a soft constraint that biases exploration without dictating behavior.

Within each episode, the high-level data flow is:

1. The environment emits the current observation o_t and exposes a task description g; the agent maintains the last p actions $a_{t-p:t-1}$.
2. A scheduling gate (parameterized by frequency k) decides whether to query the LLM at time t.

3. If scheduled, the system encodes the state into text (e.g., an ASCII grid), assembles the prompt with action history and goal, and queries the LLM to obtain a hint $\hat{h}_t$; otherwise, it sets h_t to a neutral placeholder and marks $h_{avail,t}$=0.
4. The enhanced observation o'_t is constructed and passed to the policy π_θ.
5. The agent samples an action $a_t \sim \pi_\theta(a_t \mid o'_t)$; the environment transitions to o_{t+1} and yields reward r_t.
6. Trajectories store augmented tuples for learning; policy optimization proceeds as in standard RL.

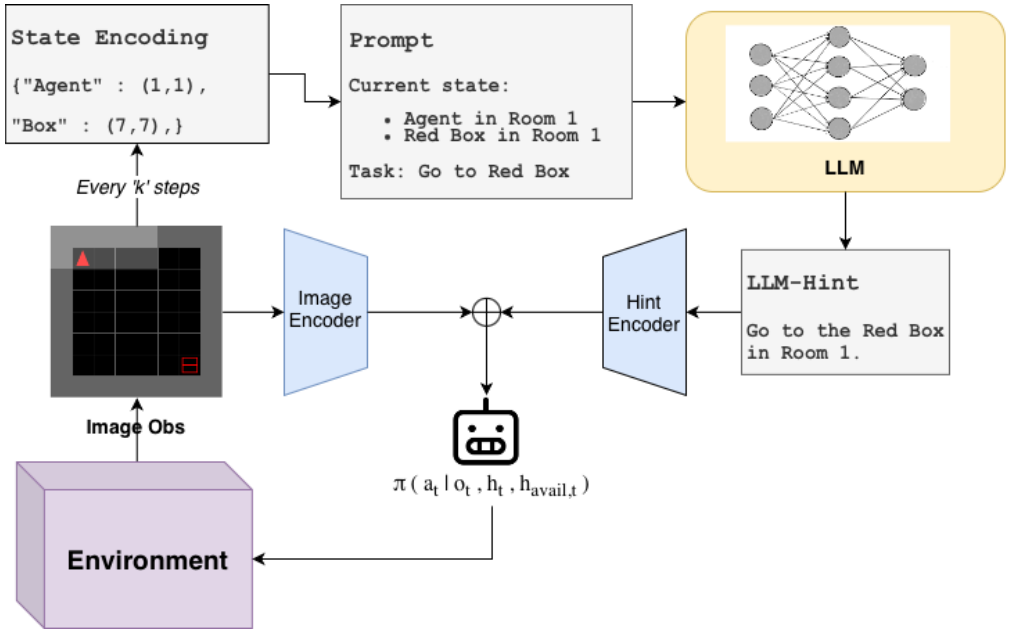


Figure 4.1: Overall pipeline for integrating LLM guidance into RL training. The diagram illustrates the core interaction: the environment emits an observation, which is encoded (with recent actions and goal) to form a prompt; the LLM returns a hint, which is incorporated into an enhanced observation for the agent. The process forms a closed decision loop, with the agent learning from both the standard observation and the LLM-provided hint.

This design leverages the LLM as a source of structured prior knowledge to accelerate exploration and reduce dithering in challenging states, while maintaining robustness when hints are imperfect. Practical concerns such as latency and cost are controlled by the query schedule (frequency k) and batching. When queries fail or are skipped, the system falls back to $h_{avail,t}$=0 without altering the training loop.

4.2 Problem Formulation

We consider partially observed Markov decision processes (POMDPs) defined by the tuple $(S, A, O, \Omega, T, \gamma, R)$, where S is the state space, A is the action space, Ω is the

observation space, $T(s' \mid s, a)$ is the transition function, $O(o \mid s, a)$ is the observation function, $R(s, a)$ is the reward function, and $\gamma \in (0, 1]$ is the discount factor. At time t, the agent receives an observation $o_t \in \Omega$ and chooses an action $a_t \in A$ according to its policy. In a goal-conditioned setting, a task description g specifies the target condition (e.g., reaching a location or interacting with an object).

The central design choice is to enhance the agent's observation with LLM-provided guidance and a binary availability flag. We define the enhanced observation as

$$o'_t = \{\, o_t,\ h_t,\ h_{avail,t} \,\}. \tag{4.1}$$

where o_t is the standard environment observation, h_t is the LLM-generated hint at time t, and $h_{avail,t} \in \{0, 1\}$ indicates whether a valid hint is currently present. When $h_{avail,t} = 0$, we use a neutral placeholder for h_t to maintain a consistent input shape.

This formulation makes the motivation for LLM integration clear: by injecting structured prior knowledge through o'_t rather than altering T or R, we bias exploration in information-poor regimes while preserving the original control problem. The agent remains the sole decision-maker and can learn to exploit or ignore hints based on their predictive value for return, ensuring robustness even when hints are noisy or occasionally misleading.

4.3 LLM Hint Generation

Hints are generated by prompting the LLM with three inputs: (i) a textual encoding of the current observation, (ii) a short history of the previous p actions, and (iii) a goal description g. We write the hint-generation function as

$$h_t = LLM\big(encode(o_t),\ a_{t-p:t-1},\ g\big), \tag{4.2}$$

where $encode(o)$ transforms the current observation into a structured text prompt (see Section 4.3.1 for details), and the sequence $a_{t-p:t-1}$ denotes the most recent p actions. We keep the encoding choice general and environment-dependent. Examples include grid-structured layouts, natural language descriptions, tuple lists of objects and coordinates, or egocentric relative descriptions.

4.3.1 State-to-Text Encoding

The encoder $encode(o_t)$ converts the current observation into a compact textual artifact suitable for LLM reasoning. For grid-like domains, a grid-structured encoding enumerates cells in a fixed-width layout, uses a small, disjoint legend (e.g., agent, walls, doors, goal objects), and encodes the agent's orientation. In non-grid settings, the encoding may list objects, relations, or feature tuples. Across encodings, we use a stable schema across timesteps to reduce prompt variance and include minimal metadata (e.g., dimensions, legend) to control token overhead.

Environment-specific encoder choices and formatting details are given in the Results for each environment (see Sections 5.2.1, 5.3.1, and 5.3.2).

4.3.2 Prompt Generation

Given $encode(o_t)$, the action history $a_{t-p:t-1}$, and goal description g, we assemble a prompt that first specifies the legend and goal, then presents the state encoding, followed

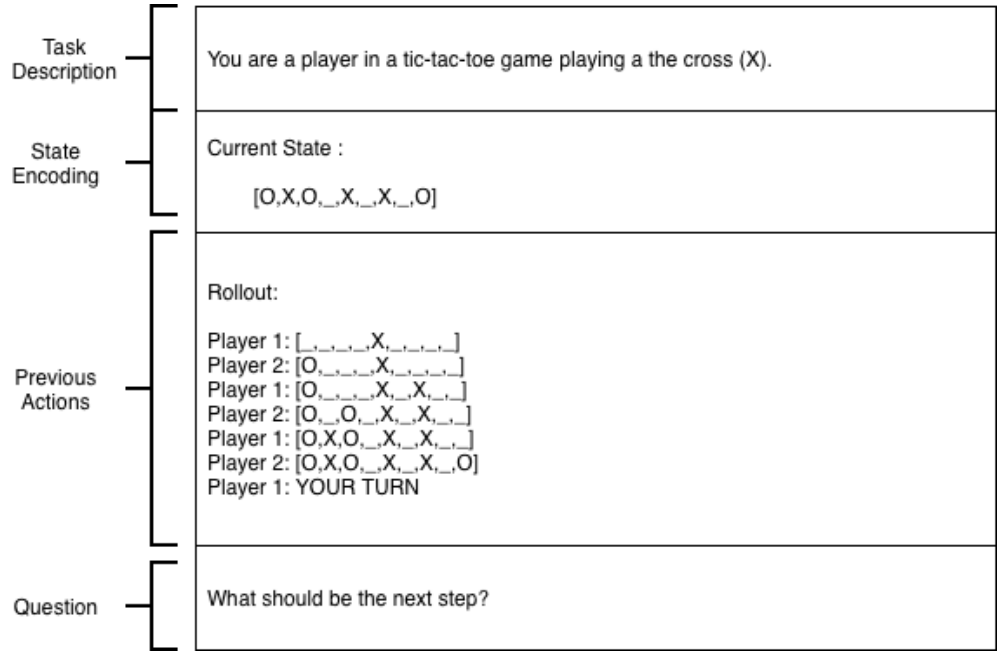


Figure 4.2: Prompt template for LLM hint generation, showing four sections: task description, state encoding, recent actions, and an explicit question to elicit the hint.

by the recent action history (see Figure 4.2). We order elements to prime the LLM with task intent before exposing the state, which empirically reduces off-task suggestions. We constrain the output format explicitly (e.g., "return a single token from the set of primitive actions") to simplify parsing and downstream alignment with the agent's discrete action space.

The action history is represented as a short, ordered list of symbolic actions $a_{t-p:t-1}$ with a fixed maximum length p (e.g., p=5). We retain only the most recent actions to provide local temporal context without inflating prompt size. When longer context is beneficial, this can be extended to a truncated trajectory history. An evaluation of different values of p is provided in Section 5.2.3.

To control latency and cost, prompts are generated only when the scheduling rule triggers a query (frequency k). When not scheduled, we skip prompt construction and set $(h_t, h_{avail,t}) \leftarrow (h^0, 0)$, ensuring a consistent input shape for the agent.

4.3.3 Prompting the LLM

At scheduled timesteps, the prompt is sent to the LLM to obtain a hint. We use a high-capacity model with chain-of-thought prompting to elicit step-wise reasoning, but we restrict the final answer to the specified schema. In practice, we throttle queries by k and batch multiple prompts where infrastructure permits. Transient failures are handled with bounded retries; if unsuccessful, the system falls back to $(h^0, 0)$ without interrupting training.

4.3.4 Parsing LLM Output

The LLM's response is parsed into the hint space $\mathcal{H}$. For primitive-action hints, we validate that the token belongs to the action set and map it to the agent's discrete action index. For subgoal or instruction outputs, we maintain a deterministic mapping to the closest admissible primitive suggestion to preserve compatibility with the policy input. Any malformed outputs trigger a safe default to h^0 with $h_{avail,t}{=}0$ and are logged for diagnostics.

Hints are provided at a fixed frequency k during training. The action history size p controls the temporal context supplied to the LLM. In our experiments, we use small values of p (e.g., $p{=}5$) that are sufficient for short-horizon dependencies while keeping prompts compact. We consider both primitive-action hints and subgoal hints; primitive-action hints yield more stable benefits in practice. Operationally, when a hint is scheduled at time t (i.e., when t is a multiple of k), the system queries the LLM and sets $h_{avail,t}{=}1$; otherwise it sets $h_{avail,t}{=}0$ and supplies a neutral placeholder for h_t.

4.4 Soft Constraint Mechanism

LLM-hints act as soft heuristics that bias exploration without dictating behavior. Conceptually, the augmented observation o'_t modifies the policy's sufficient statistics: the agent can exploit hint information when it correlates with higher return, or ignore it otherwise. This differs from hard-control designs where the LLM directly selects actions, enforces hierarchical sub-policies, or reshapes the reward function at each step.

Connections to heuristic search and policy shaping are instructive. In Go-based systems, carefully designed priors and value estimates guide search while preserving the ability to deviate when evidence accumulates [49]. Related strands in advice-taking and human-in-the-loop RL similarly provide guidance that is integrated but not determinative [35, 24, 53]. Our mechanism parallels these ideas: the hint enters as an additional input feature whose influence is learned end-to-end through gradient-based optimization, thereby preserving robustness when hints are imperfect.

4.5 RL Training with Enhanced Observations

Let π_θ denote a parameterized stochastic policy with parameters θ. Given the enhanced observation in (4.1), we define

$$\pi_\theta(a_t \mid o'_t) = \pi_\theta\big(a_t \mid o_t,\ h_t,\ h_{avail,t}\big). \tag{4.3}$$

The training objective is the standard RL goal of maximizing expected discounted return, with one change: the policy and (optionally) value function receive the augmented input o'_t. The underlying environment dynamics T and reward function R are unchanged. Hints enter only as features in the agent's input, preserving the soft-constraint property. The pipeline is algorithm-agnostic, so both policy-gradient and value-based methods can be used without altering their core optimization routines. Data are collected as augmented tuples $(o'_t, a_t, r_t, o'_{t+1}, \text{done}_t)$, enabling the agent to learn when hint information is predictive of higher return and when it should be ignored.

Algorithm 4.1 summarizes the training loop. We write `encode_state(o)` for the state encoder (grid-structured or otherwise, depending on the environment) and `neutral_action`

for the neutral hint placeholder. The hint availability h_{avail} is toggled by the frequency parameter k as described above.

Listing 4.1: Algorithm 1: LLM-Guided RL Training

```
# Initialize environment env, LLM model, RL agent pi
# Set hint frequency k, action history size p
for episode in range(1, N+1):
    o0, g = env.reset()
    a_history = []  # initialize action history
    t = 0
    done = False
    while not done:
        t = t + 1
        if t % k == 0:
            s_encoded = encode_state(o_t)
            h = LLM(s_encoded, a_history, g)
            h_avail = 1
        else:
            h = neutral_action
            h_avail = 0
        o_prime_t = {o_t, h, h_avail}  # enhanced observation
        a_t = pi(o_prime_t)            # agent action
        o_t1, r_t, done = env.step(a_t)
        # update action history (keep last p actions)
        a_history.append(a_t)
        if len(a_history) > p:
            a_history.pop(0)
        # train agent pi on (o_prime_t, a_t, r_t, o_prime_t1)
        train(pi, o_prime_t, a_t, r_t, o_t1, h, h_avail)

# End
```

Following the pseudocode, training proceeds as in standard RL. The only additions are (i) periodic LLM queries to obtain h_t and (ii) the construction of enhanced observations o'_t per Eq. (4.1). The agent's trajectory storage records augmented tuples; optimization then follows the chosen algorithm (e.g., on-policy or off-policy), letting the networks learn internal representations that capture when and how hint information should influence decisions while preserving autonomy.

4.6 Summary

This chapter presented a methodology for integrating LLM guidance into RL via enhanced observations and soft constraints. We formalized the task as a POMDP and introduced an exogenous hint process that augments the agent's observation with a hint h_t and availability flag $h_{avail,t}$, yielding the enhanced observation o'_t in Eq. (4.1). Hints are generated by encoding the current observation, recent action history, and task goal into a compact prompt, querying an LLM at a scheduled frequency, and parsing the response into a stable schema aligned with the agent's action space.

The training pipeline is algorithm-agnostic: any policy-gradient or value-based method can consume the same augmented input without modifying its core optimization routine. Because hints enter solely as input features, the environment dynamics and reward function are preserved, ensuring the guidance operates as a soft constraint that biases exploration without dictating behavior. The accompanying pseudocode summarized how hints are scheduled, injected into observations, and logged in augmented trajectories while leaving optimization to the chosen RL algorithm.

Overall, the design provides a principled, modular interface between LLM-derived priors and standard RL training. It enables agents to learn when to leverage or ignore hints based on reward feedback, improving exploration efficiency while maintaining robustness. The next chapter evaluates alternative encodings, prompting strategies, and algorithm choices, and quantifies their effects on performance and learning dynamics.

Chapter 5
Evaluation and Results

5.1 Overview

This chapter presents the empirical evaluation of the proposed framework and its alignment with the research questions (RQ1, RQ2) and corresponding hypotheses. We organize the evaluation around two complementary analyses: (i) the capacity of prompt engineering to elicit structured, state-contingent hints from an LLM, and (ii) the impact of integrating such hints as soft inputs on RL efficiency, performance, and robustness. The study includes a main evaluation in Minigrid and an extended evaluation in *TicTacToe* and *Deal or No Deal*, providing both depth and breadth. The criteria and settings below are designed to give clear, testable evidence for Hypothesis 1 and Hypothesis 2 while preparing the ground for the results reported in the subsequent sections.

5.1.1 Evaluation Criteria

The evaluation addresses two research questions and their associated hypotheses:

RQ1. How can prompt engineering encode task-specific dynamics, rules, and constraints into an LLM so that, given compact state summaries, it reliably generates structured, state-contingent action hints suitable for the agent's observation space?

RQ2. To what extent do LLM-generated hints, integrated as soft inputs in an augmented observation space, improve RL training in terms of sample efficiency, final performance, and robustness to imperfect or absent guidance compared to tabula-rasa and hierarchical baselines?

Hypothesis 1. Carefully designed prompts that encode task rules, action schemas, and compact state summaries will elicit structured, state-contingent hints from the LLM that are consistently task-relevant and suitable for direct inclusion in the agent's observation space.

Hypothesis 2. Integrating LLM-generated hints as soft inputs will improve sample efficiency and final performance over tabula-rasa baselines, while remaining robust by

learning to down-weight or ignore low-quality or absent guidance.

Criteria derived from RQ1 and Hypothesis 1. We evaluate the *quality and consistency* of LLM-generated hints using measures of structural validity (adherence to the specified schema), state-contingency (alignment with the provided state summary), consistency across prompts and seeds, and parseability. Qualitative inspection complements these measures with examples and failure modes.

Criteria derived from RQ2 and Hypothesis 2. We quantify the *impact on RL training* via learning curves, frames-to-threshold (sample efficiency), asymptotic performance, and training stability, alongside robustness evaluations under noisy or absent hints. Comparisons against tabula-rasa baselines establish relative gains and sensitivity to guidance quality.

5.1.2 Environment Choice

To balance long-horizon complexity with interpretability and computational tractability, we adopt two complementary categories of environments. *Category I (Main)* comprises Minigrid, a family of grid-based tasks with large state spaces, sparse rewards, and longer training horizons, serving as the primary setting for comprehensive analyses. *Category II (Extended)* comprises TicTacToe and Deal or No Deal, smaller two-player or decision-based domains that are highly interpretable and lower in computational cost. Together, these environments provide a diverse basis for evaluating both the generation of LLM hints and their integration within RL training. The next section presents the main results in Minigrid.

5.2 Main Results: Minigrid

This section presents the main empirical results in Minigrid, addressing RQ1 (whether prompt engineering yields reliable, state-contingent hints suitable for soft integration) and RQ2 (whether such hints improve RL training in terms of sample efficiency and final performance). Building on the methodology and definitions in Section 4, we follow the experimental protocol and terminology established in the preceding sections and present qualitative and quantitative evidence in support of the hypothesis.

5.2.1 Environment and Implementation Details

We evaluate three Minigrid (BabyAI) task variants of increasing difficulty: *GoToObj* (easy), *OpenDoor* (medium), and *PickupLoc* (hard). Minigrid comprises 2D grid-based, goal-oriented tasks with a discrete action space; the agent must navigate mazes, interact with objects (e.g., doors, keys, boxes), and solve longer-horizon problems in the harder variants. Observations and actions follow standard Minigrid definitions, and policies are trained with Proximal Policy Optimization (PPO).

Three hint configurations are considered: (i) a *Baseline* agent without hints (tabula-rasa), (ii) an *Oracle* that supplies optimal hints (upper bound), and (iii) *LLM-hints* integrated as soft inputs in an augmented observation space. We vary hint frequency $f \in \{1, 5, 10\}$, with LLM-hints evaluated at $f = 5$ and $f = 10$. For GoToObj and OpenDoor we train for

3M frames, and for PickupLoc for 5M frames. In a text-conditioned policy variant, the mission string is optionally concatenated to observations.

LLM guidance is produced by *Llama3-70b*. Prompts encode task rules, action schemas, and compact state summaries, and they employ Chain-of-Thought (CoT) reasoning unless otherwise specified. Hints are serialized into a fixed schema and concatenated into the policy's observation channel (soft integration), enabling the agent, through reward feedback, to learn when to use or ignore guidance.

(a) BabyAI environments

Environment	Description
GoToObj (easy)	Navigate to a target object in a single room with no doors or distractors; direct line-of-sight.
OpenDoor (medium)	Go to a specified door (by color/location) that is unlocked and in the current room; requires object identification and navigation.
PickupLoc (hard)	Pick up an object described by its location within a single room; requires spatial reasoning, identification, and precise navigation.

(b) Methods compared

Method	Description
Baseline	Standard PPO without hints
LLM-hints (ours)	PPO with LLM action suggestions via enhanced observations
Oracle	PPO with ground truth hints from optimal BabyAI planner

Table 5.1: Environments and methods evaluated in Minigrid. (a) BabyAI tasks included in our evaluation. (b) Methods compared in experiments.

5.2.2 Evaluating LLM Suggestions

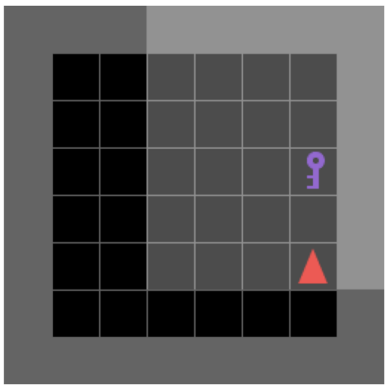
We evaluate the reliability and usefulness of LLM-generated hints under alternative prompt designs, state encodings, and output abstractions. We compare four state encodings: (i) structured natural-language descriptions; (ii) ASCII grid maps with a legend; (iii) tuple-based object lists; and (iv) relative, egocentric descriptions. We consider three output abstractions (primitive actions, subgoals, and high-level instructions) and three prompting strategies: zero-shot, few-shot, and Chain-of-Thought (CoT).

Preserving spatial structure emerges as critical. ASCII grid encodings consistently produce the highest-quality and most parseable hints across tasks. Including short action histories further improves coherence and reduces repetitive detours. Among outputs, subgoals are interpretable and semantically aligned with objectives, while primitive actions align most directly with the RL action space. A mild forward-action bias appears in primitive actions but is attenuated by CoT. Across prompting strategies, CoT reliably outperforms zero-/few-shot prompting, yielding more consistent, state-contingent suggestions. An illustrative example is shown in Figure 5.1, which visualizes the ASCII state encoding and a representative Chain-of-Thought response. Overall, RQ1 is supported: prompt-engineered task knowledge (especially with ASCII grids and CoT) elicits structured, state-contingent hints suitable for soft integration.

5.2.3 RL Training

We next assess whether integrating LLM-hints as soft inputs improves learning relative to a tabula-rasa Baseline and how these gains compare to an Oracle upper bound. We report

Environment: BabyAI-GoToObj-v0



Text Encoding: Ascii

```
ENCODING TYPE: ASCII
==================================================

Grid:
W W W W W W W W
W . . . . . . W
W . . . . . . W
W . . . . . K W
W . . . . . . W
W . . . . . ^ W
W . . . . . . W
W W W W W W W W

Legend:
^ - agent facing north
? - unseen
. - empty
W - wall
K - purple key

Mission: go to the purple key
```

(Example) LLM Chain-of-Thought Response:

```
Prediction(
  reasoning="The agent's mission is to go to the purple key.  The purple key is located
to the north of the agent.  Since the agent is facing north, it first needs to move
forward towards the purple key.  The action to move the agent forward would be the most
efficient first step.",
  primitive_action=2,
  subgoal=GoNextToSubgoal
)
```

Figure 5.1: Environment visualization with ASCII encoding (top) and corresponding LLM reasoning chain-of-thought response (bottom). The LLM analyzes the spatial relationships and mission requirements to determine the optimal action.

Representation Method	**Description**	**Example**
Natural Language Description	Free-text description of nearby objects	"In front of you is a red key."
ASCII Grid Map	Text grid with symbols and legend	See Appendix
Tuple List	Structured list of objects + coordinates	"[(red key, (1,2))]"
Relative Descriptions	Object locations relative to agent	"Red key 1 tile ahead"

Table 5.2: State encodings evaluated for LLM prompting in Minigrid (summary).

final rolling success rates and sample efficiency (frames to reach defined thresholds), and we analyze the effects of hint cadence and mission text conditioning.

End performance exhibits a ceiling effect in GoToObj ($\approx 100\%$ across all methods), moderate differences in OpenDoor, and the largest gaps in PickupLoc. In the hardest setting, LLM-hints at $f = 5$ substantially improve over the Baseline (29.5% vs. 17.2%) but remain below the Oracle upper bound (Table 5.4).

Learning curves (Figure 5.2) show accelerated early learning and improved stability, particularly in OpenDoor and PickupLoc. We quantify these gains using frames-to-threshold analyses.

Table 5.5 highlights substantial acceleration under guidance. Relative to the Baseline, Oracle $f = 1$ achieves up to $\sim 28\times$ faster attainment of mid-to-high thresholds in GoToObj and yields progress where the Baseline makes none within budget (OpenDoor at 90%; PickupLoc at 50–90%). LLM-hints approach these trends at moderate frequencies,

Output Type	Description	Example	Abstraction Level
Primitive Action	Single atomic action	"move forward" / "0"	Low
Subgoal	Short-term objective	"PickupSubgoal(red key)"	Medium
Instruction	Full temporal instruction	"Go to red key and pick it up."	High

Table 5.3: LLM output abstractions considered for hint generation.

Environment	Baseline	Oracle f=1	Oracle f=5	LLM-hints f=5	LLM-hints f=10
GoToObj	100.0%	100.0%	100.0%	**100.0%**	100.0%
OpenDoor	74.2%	100.0%	96.4%	**75.0%**	74.3%
PickupLoc	17.2%	99.4%	42.1%	**29.5%**	18.9%

Table 5.4: Final rolling win rates across environments and configurations (no text).

particularly in PickupLoc.

We also analyze the role of cadence. Figure 5.4 compares trajectories at $f \in \{5, 10\}$ for LLM-hints alongside Oracle frequencies. Across tasks, $f = 5$ generally outperforms $f = 10$, suggesting that overly frequent hints can interfere with policy learning, whereas moderate cadence preserves guidance benefits while allowing sufficient exploration.

Finally, we analyze text conditioning. Adding the mission string to the policy input yields mixed effects (Table 5.6): slight gains in OpenDoor and degradations in PickupLoc. This suggests that textual goals may help or distract depending on environment structure and the alignment between textual and spatial information.

5.2.4 Discussion

Taken together, the results support both hypotheses within Minigrid. For RQ1, prompt engineering that preserves spatial structure (ASCII grid encodings) and elicits CoT reasoning yields structured, state-contingent hints that are readily serialized into an augmented observation space. Action histories modestly improve coherence. Subgoals provide interpretable intermediate objectives, while primitive actions align most directly with the learner's control interface, mitigating abstraction mismatch at training time.

For RQ2, soft-input integration of LLM guidance improves learning dynamics relative to a tabula-rasa baseline, with the strongest gains in the most challenging setting (PickupLoc). Oracle guidance establishes clear upper bounds for both final performance and sample efficiency, demonstrating up to $\sim 28\times$ reductions in frames to threshold and converting failures into successes where the Baseline does not reach targets within budget. LLM-hints close part of this gap, especially at moderate hint frequencies ($f = 5$), while $f = 10$ is generally less effective but not counterproductive.

An important negative result concerns subgoal hints in training: despite their interpretability and usefulness for analysis, directly using subgoal outputs as hints does not improve PPO training compared to primitive-action hints. This suggests an abstraction mismatch between high-level intent and low-level control, which can dilute credit assignment signals. A representative result is shown in Figure 5.5 for BabyAI-GoToObj-v0, where oracle subgoal hints do not surpass the text-conditioned baseline.

Overall, the Minigrid experiments indicate that (i) preserving spatial structure in prompts and using CoT is critical for reliable hint quality (RQ1), and (ii) integrating primitive-action LLM hints as soft inputs improves sample efficiency and, on harder tasks, final

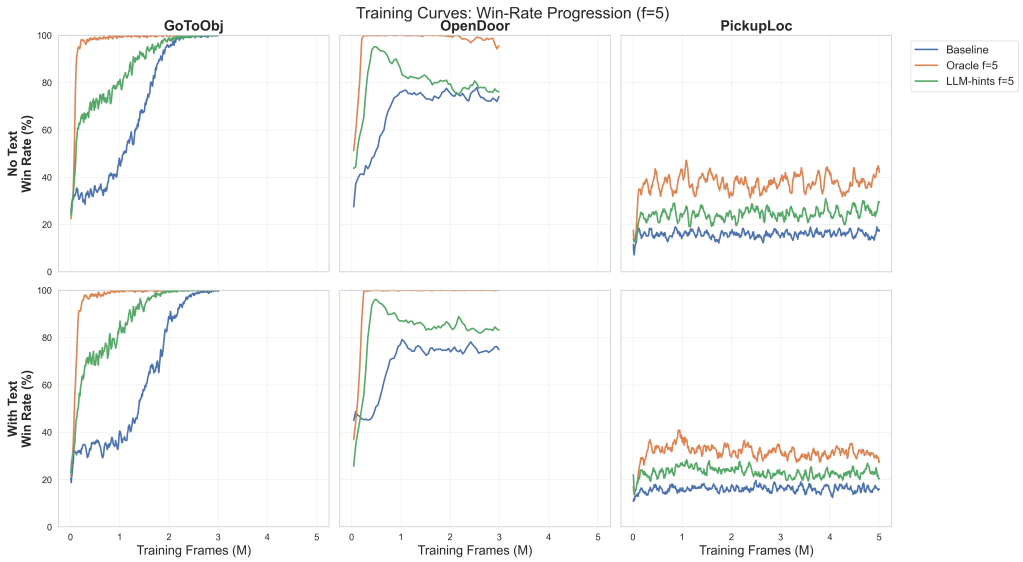


Figure 5.2: Training curves for frequency $f = 5$: Baseline, Oracle $f = 5$, and LLM-hints $f = 5$ across GoToObj, OpenDoor, and PickupLoc, with/without mission text.

Environment	Threshold	Baseline	LLM-hints f=5
GoToObj	50%	1.08M	**120K (9×)**
	75%	1.58M	**180K (8.8×)**
OpenDoor	50%	475K	**400K (1.2×)**
	75%	950K	**800K (1.2×)**
PickupLoc	25%	Never	**Achieved**

Table 5.5: Sample efficiency: frames to reach success thresholds.

performance over tabula-rasa learning (RQ2), while remaining below the oracle upper bound. Hint cadence and text conditioning are important design choices: moderate frequencies tend to work best, and textual goals should be applied judiciously.

5.3 Extended Results: TicTacToe & Deal or No Deal

This section examines generality beyond Minigrid using two compact, interpretable domains: *TicTacToe* and *Deal or No Deal*. The aim is twofold: (i) assess whether prompt designs elicit reliable, state-contingent hints in small problems, and (ii) test whether such guidance, when softly integrated, yields measurable improvements in RL. Given the short horizons and small state spaces, we emphasize LLM suggestion quality and interpretability, reporting only brief RL comparisons for completeness.

5.3.1 TicTacToe

We describe the evaluation setup and outcomes in TicTacToe, comparing agents with and without LLM guidance and relating prompt-encoded strategies to optimal play.

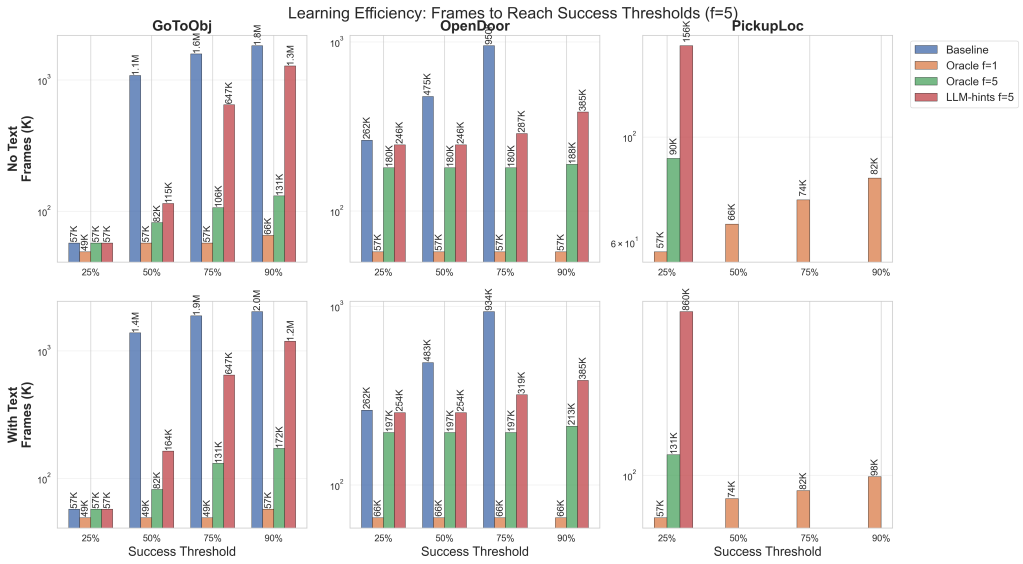


Figure 5.3: Learning efficiency at $f = 5$: frames to reach 25%, 50%, 75%, and 90% thresholds for Baseline, Oracle, and LLM-hints.

Environment	Baseline	Baseline +Text	LLM-hints f=5	LLM-hints f=5 +Text
GoToObj	100.0%	99.6%	100.0%	100.0%
OpenDoor	74.2%	75.9%	75.0%	77.9%
PickupLoc	17.2%	15.9%	29.5%	20.4%

Table 5.6: LLM-hints performance with and without mission text conditioning.

Environment and Implementation Details

TicTacToe (TTT) is a small, fully solvable, two-player game on a 3×3 grid. We include it as an extended evaluation to probe the quality of LLM-generated hints in a compact domain where optimal play is well understood and performance ceilings are high. The aim here is not to chase asymptotic gains over a near-optimal baseline, but to analyze how prompt engineering, representation choices, and action masking affect hint validity, state-contingency, and interpretability.

Training uses the PettingZoo `tictactoe_v3` environment with a masked 9-action space. Following the DQN implementation summarized earlier, observations are encoded as a 36-dimensional vector (27 board features from per-cell triplets plus a 9-dimensional action mask), and hints can augment the input by a 10-dimensional vector (a 1-bit availability flag and a 9-dimensional one-hot of the suggested move). The agent is a multilayer perceptron (36→128→9) trained with Adam at learning rate 5×10^{-5} and discount $\gamma = 1.0$; invalid actions are masked prior to the argmax. The step budget is intentionally modest, and opponents include random, self-play, and simple heuristics, reflecting the short horizons and small state space of TTT.

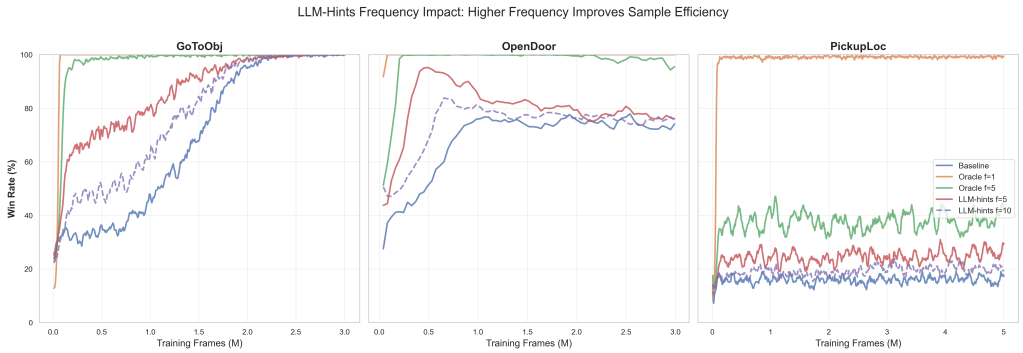


Figure 5.4: Frequency comparison: Baseline, Oracle ($f = 1, 5$), and LLM-hints ($f = 5, 10$) across environments.

Evaluating LLM Suggestions

We first assess whether prompt-engineered guidance yields valid and state-contingent action hints that are suitable for soft integration. Three findings emerge.

First, the structure of the prompt is decisive. Clear task descriptions and explicit input–output schemas substantially reduce invalid outputs. In contrast, removing the task description (while keeping state and format) increases invalidity, suggesting that even for a familiar game, explicit task framing anchors the model's output space. By comparison, the choice of board representation (1D vector versus 2D matrix; symbolic versus numeric markers) has negligible influence on validity when the task description is present, indicating representational robustness.

Second, providing the set of available actions is critical. Across models and seeds, including the action mask reduces both the fraction of invalid episodes and the percentage of invalid moves (Table 5.8). This aligns with Minigrid (Section 5.2): constraining the output space through structured inputs improves adherence to the schema and decreases off-support predictions, without otherwise inflating the prompt.

Third, qualitative analysis reveals sensible priors coupled with occasional lapses. Models show a strong center preference, perform tactical blocks, and sometimes set up forks; yet they also miss obvious wins or allow preventable losses. Figure 5.6 illustrates both behaviors for Llama3-70B. These observations are consistent with the idea that a well-specified prompt and mask elicit high-quality, parseable hints that align with board context, while residual mistakes reflect limits in local tactical consistency.

To quantify alignment with strong references, we compare LLM suggestions against two ground truths: (i) a DQN policy trained in the same environment and (ii) an oracle (minimax) policy. We compute match rates under two criteria: symmetric-match (crediting action symmetries of the board) and exact-match. Results (Table 5.7) show that larger models align more frequently with both DQN and oracle moves, with a sizable gap between symmetric and exact matches that reflects the role of board symmetries in TTT. The higher alignment with DQN than with the oracle for some models suggests that prompts and examples may anchor the LLM toward competent—but not necessarily optimal—play.

Taken together, these analyses support RQ1 in a small domain: prompt engineering that makes the task explicit and constrains the output space (via the available-action

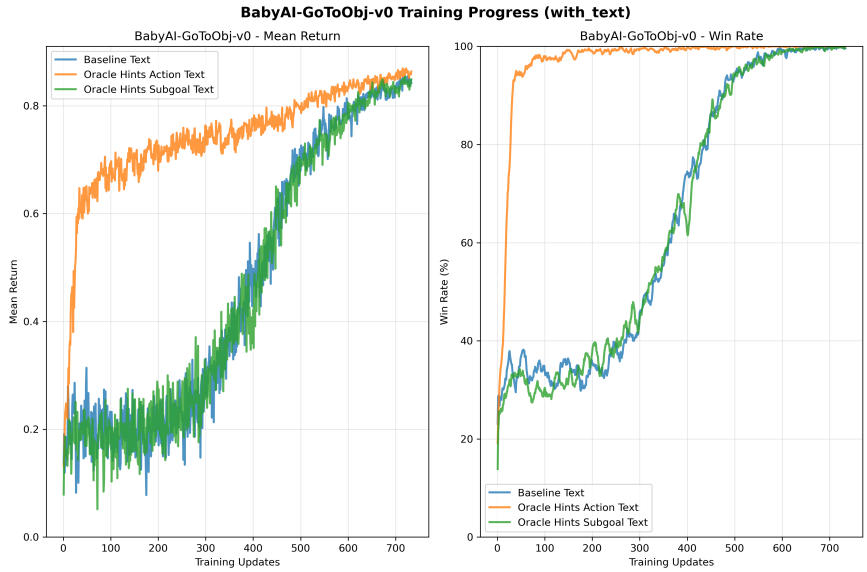


Figure 5.5: Subgoal-based hints (oracle) versus text-conditioned and baseline agents in GoToObj with text. Subgoal hints do not improve PPO training.

LLM	GT-DQN match (%)	GT-Oracle match (%)
Llama3.2-3B	12 (10 exact)	9 (9 exact)
Llama2-13B	13 (11 exact)	16 (15 exact)
Llama3-70B	57 (42 exact)	40 (39 exact)

Table 5.7: Alignment of LLM suggestions with ground truths in TTT. We report symmetric-match rates (parentheses: exact-match) over 100 suggestions, using Chain-of-Thought prompting. Symmetric matching credits board symmetries; exact matching requires the identical move.

mask) elicits structured, state-aware hints. The ground-truth comparison corroborates that larger models produce suggestions closer to strong references, while qualitative examples expose the remaining tactical gaps.

RL Training and Limitations

We next examine whether these hints, integrated as soft inputs, improve DQN training. We evaluate two encodings—one-hot 9-dimensional versus integer index—for incorporating a single suggested move alongside the 36-dimensional observation. In this small, short-horizon domain, improvements over a strong no-hint baseline are limited: both encodings yield similar learning curves and final returns (Figure 5.7). We attribute the modest gains to three factors. First, the effective state space is near-tabular with a shallow credit-assignment structure; given action masking and simple curricula, DQN rapidly attains competent play even without hints. Second, the horizon is extremely short, so the potential to propagate guidance across time is minimal, reducing the value of occasional soft inputs. Third, the performance ceiling is high under standard training, leaving little headroom for further acceleration.

We also explored hint cadence (omitted for brevity), observing that very frequent sugges-

LLM Configuration	LLM	Invalid Episodes (%)	Invalid Moves (%)
Without Available Actions	Llama3.2-3B	100	40.00
	Llama2-13B	100	47.56
	Llama3-70B	70	24.13
With Available Actions	Llama3.2-3B	10	3.22
	Llama2-13B	10	3.03
	Llama3-70B	0	0.00

Table 5.8: Effect of providing available actions in the prompt on validity. Ten evaluation games (≈30 suggestions). Prompting: Chain-of-Thought.

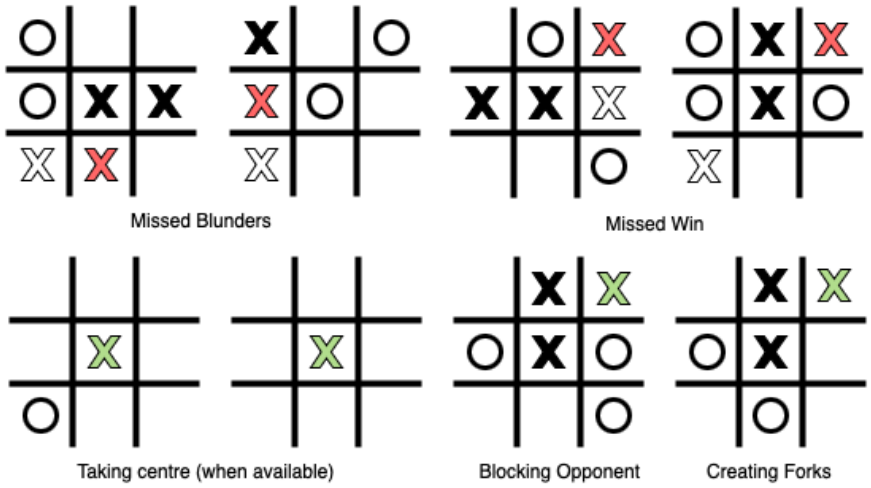


Figure 5.6: Illustrative LLM suggestions (Llama3-70B): examples of correct tactical play (center preference, blocks) and failure cases.

tions can interfere with autonomous learning, while moderate or sparse hints neither harm nor substantially accelerate convergence. This pattern mirrors Minigrid: when the environment is simple, the marginal value of frequent guidance diminishes, and policies learn to ignore redundant signals.

5.3.2 Deal or No Deal

Environment and Implementation Details

Deal-or-No-Deal (DoND) is a two-agent, multi-issue bargaining task derived from [31]. In each episode, the environment samples item counts and private utility vectors for both agents, after which the players negotiate for up to ten turns using a discrete set of dialogue acts (PROPOSE, INSIST, AGREE, DISAGREE, END). We include DoND as an extended evaluation because it isolates preference reasoning under uncertainty and requires forming contextually appropriate offers, yet remains computationally lightweight. The goal here is to examine whether prompt-engineered guidance captures such reasoning and whether those hints, when softly integrated into the observation space, translate into improved learning dynamics.

Implementation follows the summary in the Appendix. Hints are requested at reset and

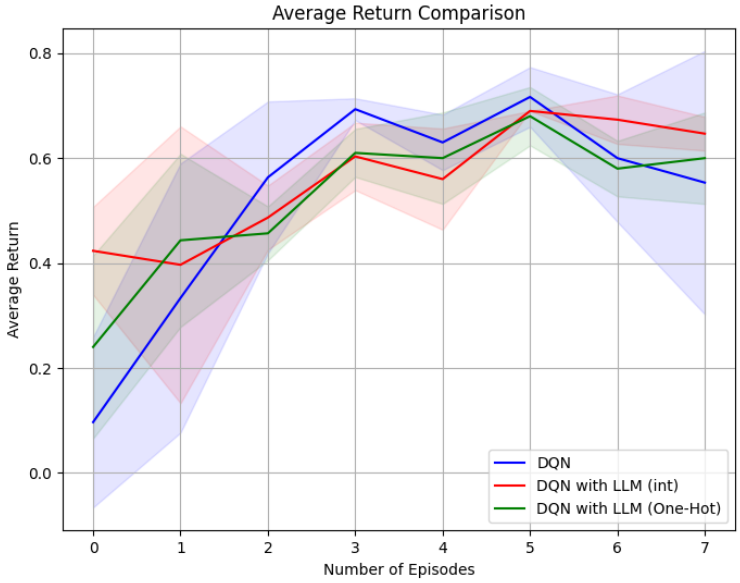


Figure 5.7: RL with LLM hints: comparison of hint encodings (one-hot vs. integer). The x-axis shows episodes (in hundreds). Differences are small in TTT due to short horizons and a small state space.

then every k steps through a wrapper that presents the current counts, the agent's utilities, the last partner act and offer, remaining turns, and per-dimension caps. The prompt enforces a strict single-line JSON response with keys `act_type`, `oA` (three integers), and `confidence`. Returned hints are validated against legality constraints and mapped to a 10-dimensional feature vector—an act one-hot (5), a normalized allocation proposal (3), a confidence scalar (1), and a hint-availability bit (1)—which is concatenated to the policy input for soft integration.

Evaluating LLM Suggestions

We first evaluate whether the prompt elicits structured and state-contingent outputs suitable for direct serialization. The template enumerates the action space, exposes recent interaction context (last act and offer), and specifies remaining turns and per-dimension caps; the model must return a single JSON object. Under this configuration, the LLM's suggestions match a ground-truth dataset in approximately **85%** of cases over $n = 100$ sampled contexts. Parsing reliability was high due to the single-line JSON constraint and explicit legality checks.

Metric	Value
Hint correctness vs. dataset	85%
Samples evaluated	100

Table 5.9: Quality of LLM suggestions in DoND using the strict JSON prompt.

Qualitative inspection further suggests that the model constructs coherent offers and

employs `AGREE` or `INSIST` when the partner's proposal meets the agent's utility threshold, while reserving counter-proposals when trade-offs are unfavorable. Errors typically arise when the last partner act is under-weighted or when multiple near-ties exist among feasible allocations. Taken together, these results indicate that prompt structure and explicit state fields produce reliable, parseable, and analytically useful hints, supporting RQ1.

RL Training and Limitations

We then examine whether these hints alter downstream learning. PPO and REINFORCE agents consume the concatenated base and hint features; we compare no-hint baselines to random, expert, and LLM guidance. Learning curves in Figures 5.8 and 5.9 reveal *limited* improvements from LLM hints. Consistent with TicTacToe (Section 5.3.1), the short horizon, compact state space, and competent heuristic partner constrain the headroom for sample-efficiency gains. Excessively frequent hints can be neutral or mildly distracting, whereas moderate cadence performs on par with the no-hint baseline.

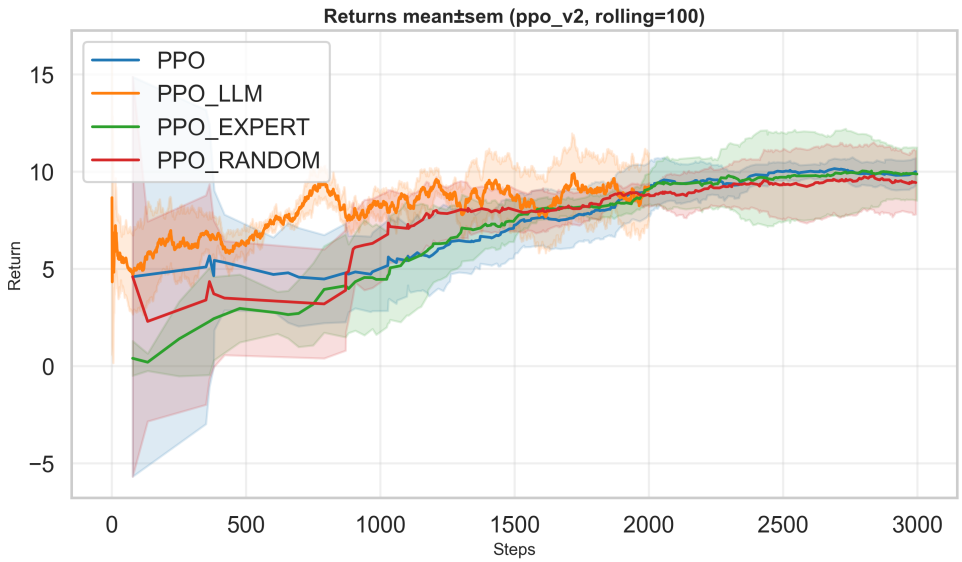


Figure 5.8: DoND: PPO variants, Baseline, LLM, Expert, and Random hints (100-episode rolling average).

5.3.3 Discussion

Across the two extended domains, a consistent pattern emerges that complements the Minigrid findings. First, for **RQ1**, carefully engineered prompts that expose salient state variables and impose a strict output schema reliably elicit structured, state-contingent hints. In TicTacToe, action masking and explicit task framing reduce invalid outputs and align suggestions with strong references; in DoND, the single-line JSON format coupled with legality checks yields high parsing reliability and 85% agreement with a ground-truth dataset. These results indicate that prompt structure and constrained output spaces are central to extracting usable guidance from LLMs in small domains.

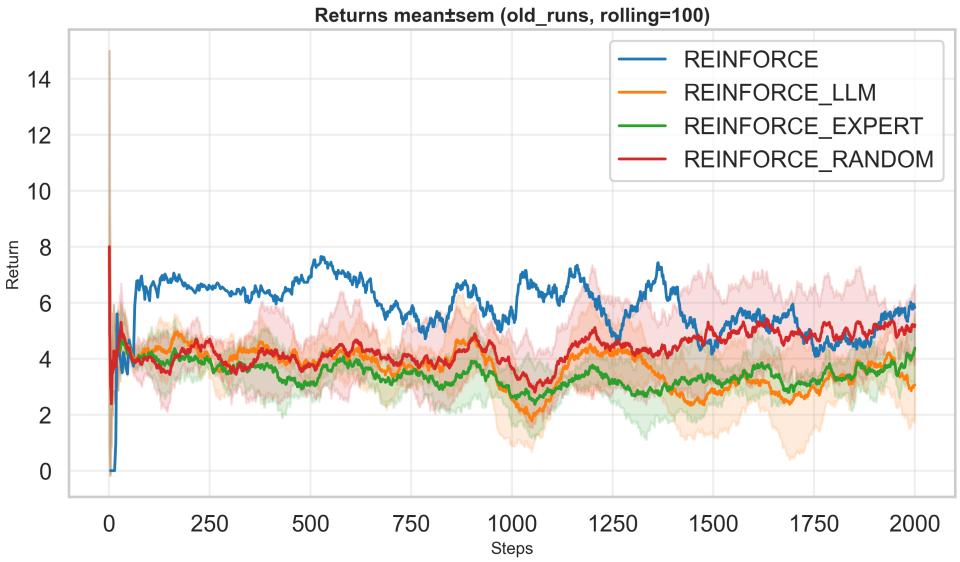


Figure 5.9: DoND: REINFORCE variants, Baseline, LLM, Expert, and Random hints (100-episode rolling average).

Second, for **RQ2**, the *downstream* impact on RL learning is modest in both environments. Short horizons, compact state spaces, and competent baselines leave limited headroom for improvements in sample efficiency or asymptotic performance. Very frequent hints can even distract learning, whereas moderate cadences are largely neutral. Thus, in compact problems the principal benefit of LLM guidance lies in interpretability and controllability, producing transparent, auditable suggestions that can be inspected, debugged, or selectively trusted, rather than in accelerating training.

Taken together with Minigrid, these observations suggest a dependency on problem scale and structure. As complexity and partial observability increase, preserving structure in prompts and injecting occasional, high-quality hints can affect learning dynamics; in near-solved domains, however, careful prompting mainly ensures reliability and analysis value without substantial training gains. This informs where LLM-guided RL is most likely to provide practical returns.

5.4 Summary

This chapter examined whether prompt-engineered guidance can be made reliable and useful for reinforcement learning, and under what conditions it translates into measurable improvements. Across domains, the findings provide clear support for **RQ1** and partial, environment-dependent support for **RQ2**.

Regarding **RQ1** (Hypothesis 1), carefully designed prompts that expose salient state variables and enforce a strict output schema consistently elicited structured, state-contingent hints. In Minigrid, preserving spatial structure (ASCII encodings) and eliciting Chain-of-Thought reasoning produced high-quality suggestions suitable for direct serialization. In the extended domains, explicit task framing and constrained outputs proved decisive: TicTacToe benefited from action masking and schema clarity, while Deal or No Deal

achieved high parsing reliability and approximately 85% agreement with a ground-truth dataset under a single-line JSON constraint. Collectively, these results confirm Hypothesis 1.

For **RQ2** (Hypothesis 2), Minigrid validated the full pipeline: integrating LLM hints as soft inputs improved sample efficiency and, on the hardest task, final performance relative to a tabula-rasa baseline, while remaining below an oracle upper bound. By contrast, in TicTacToe and Deal or No Deal, learning gains were limited. Short horizons, compact state spaces, and strong baselines compressed the potential value of external guidance. Frequent hints were neutral or mildly distracting, whereas moderate cadences were primarily benign. These outcomes partially support Hypothesis 2: LLM guidance can accelerate learning, but its impact scales with problem complexity and representational structure.

Taken together, the results delineate the strengths and limits of LLM-guided RL. Prompt engineering is a robust means of extracting reliable, interpretable decision suggestions across heterogeneous environments. However, the downstream RL benefits depend on scale and structure: as tasks grow in complexity and partial observability, occasional, high-quality hints become more consequential; in near-solved domains, their main value is transparency and controllability rather than acceleration.

These insights motivate the next chapter's discussion of design principles and future directions. We consider when and how to deploy LLM guidance, how to adapt cadence and abstraction to task demands, and how to extend the framework to richer settings where the benefits observed in Minigrid are likely to be most pronounced.

Chapter 6
Conclusion

6.1 Summary and Contributions

This thesis investigated how LLMs can provide actionable guidance to RL agents through prompt engineering and soft integration. We posed two research questions. **RQ1** asked how task-specific dynamics, rules, and constraints can be encoded into an LLM so that it reliably produces structured, state-contingent action hints. **RQ2** asked to what extent such hints, when concatenated as soft inputs within an augmented observation space, improve RL training in terms of sample efficiency, final performance, and robustness.

The work makes two conceptual contributions. First, it advances a prompting methodology that translates environment state into compact, schema-constrained inputs and compels a single, parseable output. Preserving salient structure (spatial layout in Minigrid, action availability in TicTacToe, and legality constraints in Deal or No Deal) proved essential to eliciting consistent, context-relevant suggestions. Second, it frames LLM-generated hints as *soft constraints* that enter the agent's observation space rather than as hard rules. This design lets the policy learn, through reward feedback, when to exploit or ignore guidance, thereby avoiding brittleness to occasional mistakes.

Empirically, we validated the approach across three domains of increasing diversity. In Minigrid, prompts that preserve spatial structure and elicit Chain-of-Thought reasoning yield high-quality hints that, when integrated as soft inputs, improve sample efficiency and, on harder tasks, final performance over a tabula-rasa baseline while remaining below an oracle upper bound. In TicTacToe and Deal or No Deal, the framework generalizes: prompts produce reliable, interpretable suggestions (for example, validity gains under action masking in TicTacToe and approximately 85% dataset agreement in Deal or No Deal), while downstream performance gains are limited by short horizons and compact state spaces. Collectively, these findings confirm **Hypothesis 1** and provide environment-dependent support for **Hypothesis 2**: LLM hints are reliable and useful, and their training impact scales with problem complexity and representational structure.

6.2 Limitations

The primary practical limitation of the proposed framework is the computational cost of frequent LLM queries. Although the soft-input design improves sample efficiency, it increases the inference overhead associated with frequent LLM-hint queries, particularly in long-horizon or high-throughput settings. This constraint limits scalability and may discourage exhaustive sweeps over prompt variants, cadences, and ablations.

Several avenues can alleviate these costs. Fine-tuning the underlying models on domain-specific data can improve consistency and reduce prompt length. Knowledge distillation from the LLM into lightweight student policies can retain the benefits of guidance while amortizing inference. Quantization and optimized serving further reduce latency and cost. At the RL level, adaptive hint scheduling (modulating frequency as a function of learning progress, uncertainty, or state novelty) offers a principled way to deploy guidance where it is most valuable. Finally, our evaluations show that benefits depend on environment class: substantial gains appear in complex, spatially structured tasks, whereas improvements are modest in compact, near-solved domains. This heterogeneity should inform deployment decisions.

6.3 Future Work

Future research can extend the framework along several complementary axes. Scaling to domains that require long-term reasoning and high-level planning (such as Minecraft, TextWorld, or procedural control benchmarks) would stress-test the method where soft guidance is likely to be most consequential. Beyond primitive actions, mid-level abstractions such as subgoals, partial trajectories, or option-like directives could support hierarchical learning while preserving credit assignment. A promising direction is to co-train or fine-tune open-weight LLMs for tighter task grounding, improved determinism, and reduced cost, enabling on-device or mixed on-device/cloud execution. Evaluations in more realistic settings, including robotics and embodied simulators with partial observability and safety constraints, would further assess robustness and transfer.

6.3.1 Closing Reflection

This thesis contributes to a growing body of work at the interface of language and control. By demonstrating that structured prompting reliably elicits state-contingent guidance and that soft integration can accelerate learning in sufficiently complex tasks, we take a step toward agentic AI systems that combine reasoning, perception, and action. The results also delineate limits: in compact domains, the principal value of LLM involvement lies in interpretability and controllability rather than dramatic training gains. Moving forward, the combination of scalable prompting, cost-aware deployment, and hierarchical abstractions offers a promising route to agents that make human-aligned decisions and adapt their problem-solving strategies to the demands of the environment. While technical challenges remain, the conceptual payoff, bridging language-based reasoning with embodied competence, marks a meaningful direction for future AI research.

Bibliography

[1] David Abel, Isaiah Barnett, Javier García, Andrés Londoño, Felipe Ramírez, Matthew Roberts, Valentin Schellaert, Matthias Scheutz, Stefanie Tellex, and Aitor Arrieta. 2023. The Role of Explanations in Human-in-the-Loop Reinforcement Learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 33. 560–568.

[2] Rishabh Agarwal, Akshay Krishnamurthy, John Langford, Robert Luo, and Robert E Schapire. 2019. Learning to generalize from sparse and underspecified rewards. *arXiv preprint arXiv:1902.02183* (2019).

[3] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, and others. 2022. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691* (2022).

[4] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mane. 2016. Concrete Problems in AI Safety. *arXiv preprint arXiv:1606.06565* (2016).

[5] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. A brief survey of deep reinforcement learning. *IEEE Signal Processing Magazine* 34, 6 (2017), 26–38.

[6] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, and others. 2022. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862* (2022).

[7] Emily M Bender and Alexander Koller. 2020. Climbing towards NLU: On meaning, form, and understanding in the age of data. *arXiv preprint arXiv:2005.04608* (2020).

[8] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Giri Sastry, Amanda Askell, and others. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, Vol. 33. 1877–1901.

[9] Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. 2023. Grounding large language models in interactive environments with online reinforcement learning. In *International Conference on Machine Learning*. PMLR, 3676–3713.

[10] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. *Advances in Neural Information Processing Systems* 30 (2017).

[11] Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, Pieter Abbeel, Alessandro Lazaric, and Sanmi Koyejo. 2023. Guiding reinforcement learning with large language models. *arXiv preprint arXiv:2302.02246* (2023).

[12] Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles Isbell, and Andrea Thomaz. 2013. Policy shaping: Integrating human feedback with reinforcement learning. In *Advances in Neural Information Processing Systems*, Vol. 26.

[13] David Ha and J"urgen Schmidhuber. 2018. World models. *arXiv preprint arXiv:1803.10122* (2018).

[14] Danijar Hafner. 2021. Benchmarking the spectrum of agent capabilities. *arXiv preprint arXiv:2109.06780* (2021).

[15] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. 2019. Dream to control: Learning behaviors by latent imagination. In *International Conference on Machine Learning*. PMLR, 2638–2647.

[16] Luke Hollis. 2025. Leveraging LLMs for Guiding Exploration in Reinforcement Learning. *Unpublished manuscript* (2025). Represents the core work of this thesis.

[17] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207* (2022).

[18] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Yejin Bang, Andrea Madotto, and Pascale Fung. 2023. A survey of hallucination in natural language generation. *Comput. Surveys* 55, 12 (2023), 1–38.

[19] Haobin Jiang, Junpeng Yue, Hao Luo, Ziluo Ding, and Zongqing Lu. 2024. Reinforcement Learning Friendly Vision-Language Model for Minecraft. In *European Conference on Computer Vision (ECCV)*.

[20] Dan Jurafsky and James H Martin. 2000. *Speech and language processing*. Prentice Hall.

[21] Daniel Kahneman. 2011. *Thinking, fast and slow*. Farrar, Straus and Giroux.

[22] Subbarao Kambhampati, Karthik Valmeekam, L. Guan, Kaya Stechly, Mudit Verma, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. 2024. LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks. *ArXiv* abs/2402.01817 (2024). DOI: http://dx.doi.org/10.48550/arXiv.2402.01817

[23] Robert Kirk, Eugene Vinitsky, Adam Gleave, Mikayel Samvelyan, and Tim Rockt"aschel. 2023. A survey of generalisation in deep reinforcement learning. *Foundations and Trends in Machine Learning* 16, 4 (2023), 436–622.

[24] W. Bradley Knox and Peter Stone. 2009. Interactively shaping agents via human reinforcement: The TAMER framework. In *Proceedings of the Fifth International Conference on Knowledge Capture*. 1–8.

[25] Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. *Machine Learning: ECML 2006: 14th European Conference on Machine Learning, Berlin, Germany, September 18-22, 2006, Proceedings 14* (2006), 282–293.

[26] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, Vol. 35. 22199–22213.

[27] Gregory Kuhlmann, Peter Stone, Raymond Mooney, and Jude Shavlik. 2004. Guiding a reinforcement learner with natural language advice: Initial results in RoboCup soccer. *The AAAI-04 Workshop on Supervisory Control of Learning and Adaptive Systems* (2004).

[28] Minae Kwon, Amy Yuan, Michael Janner, Chelsea Finn, and Sergey Levine. 2023. Reward design with language models. *arXiv preprint arXiv:2303.00001* (2023).

[29] Pawel Ladosz, Jung-Sheng Mo, Abdullah Al-Dujaili, Andreea Tjandra, and Tian Chen. 2022. Exploration in deep reinforcement learning: A survey. *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[30] Harrison Lee, Sam Phatale, August Pritzel, Vola Dalibard, Melina Tsimpoukelli, Laurent Sifre, Shane Legg, Oriol Vinyals, and Kevin Clark. 2023. RLAIF: Scaling reinforcement learning from human feedback with AI feedback. *arXiv preprint arXiv:2309.00267* (2023).

[31] Mike Lewis, Denis Yarats, Yann N. Dauphin, Devi Parikh, and Dhruv Batra. 2017. Deal or No Deal? End-to-End Learning for Negotiation Dialogues. *CoRR* abs/1706.05125 (2017). `http://arxiv.org/abs/1706.05125`

[32] Shuang Li, Xavier Puig, Chris Paxton, Yilun Du, Clinton Wang, Linxi Fan, Tao Chen, De-An Huang, Ekin Aky"urek, Anima Anandkumar, and others. 2022. Pre-trained language models for interactive decision-making. *Advances in Neural Information Processing Systems* 35 (2022), 31199–31212.

[33] Michael L Littman. 2015. *Reinforcement Learning: A Tutorial*. Citeseer.

[34] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931* (2023).

[35] Richard Maclin and Jude W. Shavlik. 1996. Creating advice-taking reinforcement learners. In *Machine Learning Proceedings 1996*. Elsevier, 321–329.

[36] Christopher D. Manning and Hinrich Sch"utze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press.

[37] Vincent Micheli, Eloi Alonso, and François Fleuret. 2022. Transformers are sample-efficient world models. *arXiv preprint arXiv:2209.00588* (2022).

[38] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[39] Ida Momennejad, Felix F. Wong, Demis Hassabis, and Nathaniel D. Daw. 2023. Evaluating cognitive maps and planning in large language models with virtual navigation. *arXiv preprint arXiv:2305.04742* (2023).

[40] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. 2018. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, Vol. 31.

[41] Andrew Y. Ng, Daishi Harada, and Stuart Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, Vol. 99. 278–287.

[42] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and others. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems* 35 (2022), 27730–27744.

[43] Peter C. Pang, Misha Seo, Kuang-Huei Lee, Michael Ahn, and Chelsea Finn. 2024. Knowledgeable agents from language model rollouts. *arXiv preprint arXiv:2402.15214* (2024).

[44] Shubham Pateria, Budhitama Subagdja, Ah-Hwee Tan, and Gang Chai. 2021. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)* 54, 5 (2021), 1–35.

[45] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. 2017. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*. PMLR, 2778–2787.

[46] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. 2018. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 8494–8502.

[47] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[48] Misha Seo, Peter C. Pang, Kuang-Huei Lee, Michael Ahn, and Chelsea Finn. 2024. Language models as simulators. *arXiv preprint arXiv:2402.15214* (2024).

[49] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, and others. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[50] Chan Hee Song, Jiaman Wu, Clayton Washington, Harshit Satija, He He, and Dorsa Sadigh. 2022. LLM-Planner: Few-shot grounded planning for embodied agents with large language models. *arXiv preprint arXiv:2212.04088* (2022).

[51] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press.

[52] Richard S. Sutton, Doina Precup, and Satinder Singh. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112, 1-2 (1999), 181–211.

[53] Lisa Torrey and Matthew E. Taylor. 2013. Teaching on a budget: Agents advising agents in reinforcement learning. *Autonomous Agents and Multi-Agent Systems* 27 (2013), 117–132.

[54] Alexander Trott, Stephan Zheng, Caiming Xiong, and Richard Socher. 2019. Keeping your distance: Solving sparse reward tasks using self-balancing shaped rewards. In *Advances in Neural Information Processing Systems*, Vol. 32.

[55] Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2023. On the planning abilities of large language models (a critical survey). *arXiv preprint arXiv:2302.06706* (2023).

[56] Karthik Valmeekam, Sarath Sreedharan, Alberto Olmo, and Subbarao Kambhampati. 2022. Large language models are not zero-shot planners. *arXiv preprint arXiv:2205.10625* (2022).

[57] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, Vol. 30.

[58] Alexander S. Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. 2017. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161* (2017).

[59] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, William Fedus, Aakanksha Chowdhery, Sharan Narang, and others. 2022a. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682* (2022).

[60] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2022b. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903* (2022).

[61] Joseph Weizenbaum. 1966. ELIZA—a computer program for the study of natural language communication between man and machine. *Commun. ACM* 9, 1 (1966), 36–45.

[62] Terry Winograd. 1972. *Understanding Natural Language*. Elsevier.

[63] Christian Wirth, Riad Akrour, Gerhard Neumann, and Jan Peters. 2017. A survey of learning from demonstration. *Robotics and Autonomous Systems* 93 (2017), 1–16.

[64] Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. 2021. Improving sample efficiency in model-free reinforcement learning from images. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 10674–10681.

[65] Yang Yu. 2018. Towards sample efficient reinforcement learning for StarCraft. *arXiv preprint arXiv:1812.02873* (2018).

[66] Andy Zhang, Lawrence Chan, and Pieter Abbeel. 2023a. Large language models are effective text-based policy learners. *arXiv preprint arXiv:2305.14828* (2023).

[67] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Ting Fu, Xinting Chen, Jian Zhao, Yue Zhang, and Shuming Liu. 2023b. Siren's call in the AI ocean: a survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219* (2023).

[68] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, and others. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* (2023).