CAUSAL ROBUSTNESS IN LLM GUIDED REINFORCEMENT LEARNING AGENTS

19Z720 - PROJECT WORK - I

Anandkumar NS (22z209)
Dhakkshin S R (22z215)
Kishoreadhith V (22z232)
M Raj Ragavender (22z233)
Rithvik K (22z253)

Dissertation submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING

Branch: COMPUTER SCIENCE AND ENGINEERING

of Anna University

November 2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
PSG COLLEGE OF TECHNOLOGY
(Autonomous Institution)
COIMBATORE – 641 004

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641 004

CAUSAL ROBUSTNESS IN LLM GUIDED REINFORCEMENT LEARNING AGENTS

Bona fide record of work done by

Anandkumar NS - (22z209)
Dhakkshin S R - (22z215)
Kishoreadhith V - (22z232)
M Raj Ragavender - (22z233)
Rithvik K - (22z253)

Dissertation submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING

Branch: COMPUTER SCIENCE AND ENGINEERING

of Anna University

November 2025

...……	……..……
Dr. Arul Anand N	**Dr. Sudha Sadasivam G**
Faculty guide	Head of the Department

Certified that the candidate was examined in the viva-voce examination held on ……...

……..	……..
(Internal Examiner)	(Panel member)

CERTIFICATE

Certified that this report titled **"CAUSAL ROBUSTNESS IN LLM GUIDED REINFORCEMENT LEARNING AGENTS"** for the Project Work I (19Z720) is a bonafide work of **Anandkumar NS (22z209), Dhakkshin S R (22z215), Kishoreadhith V (22z232), M Raj Ragavender (22z233), and Rithvik K (22z253)** who have carried out the work under my supervision for the partial fulfillment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science and Engineering. Certified further that to the best of my knowledge and belief, the work reported herein does not form part of any other thesis or dissertation based on which a degree or an award was conferred on an earlier occasion.

Place: Coimbatore
Date:

Members

Anandkumar NS
Dhakkshin S R
Kishoreadhith V
M Raj Ragavender
Rithvik K

Faculty Guide

Dr. Arul Anand N
Department of Computer Science and Engineering
PSG College of Technology
Coimbatore-641004

COUNTERSIGNED

Dr.Sudha Sadasivam G,
Head of the department,
Department of Computer Science and Engineering,
PSG College of Technology,
Coimbatore-641004

ACKNOWLEDGEMENT

We would like to thank the management of PSG College of Technology for providing us the infrastructure and helping us envision new ideas. We would also like to express our heartfelt gratitude to our Principal **Dr.K.Prakasan** for bestowing us with this valuable opportunity to work in our area of interest.

We also extend our sincere thanks to our Head of the Department **Dr. Sudha Sadasivam G**, Department of Computer Science and Engineering for constantly supporting us to develop our ideas and present our project to the faculty committee as a part of our partial fulfilment of the requirements leading to the awarding of B.E. degree.

We take immense pleasure in thanking our project guide and programme co-ordinator, **Dr. Arul Anand N**, Professor, Department of Computer Science and Engineering, for being a pillar of support, without whose guidance, unparalleled cooperation and constructive criticisms, this project wouldn't have been fruitful.

A sincere thanks to all the Panel Members for reviewing our project and all the Faculty Members and Staffs of the department for offering us the required support during the course of the project.

We express our thanks and gratitude to our tutor and faculty in-charge for the course *19Z720 – Project Work I*, **Ms. Anisha C.D**, Assistant Professor Department of Computer Science and Engineering for help and encouragement in the duration of the course.

Finally, we would like to thank all our student colleagues and staff members in the Computer Science Department, without whom this dissertation work would not have been completed successfully.

CONTENTS

SYNOPSIS

The usage of Large language models (LLMs) as high-level planners for reinforcement learning (RL) agents in complex real-time environments have been explored recently by a few works. This approach has some reliability concerns as LLM-generated advice can be unreliable, especially under corrupted or ambiguous world states and descriptions which in turn realizes security concerns about the decisions made following LLM advice. This work is an attempt to address this gap by proposing a causal-inference based framework to detect and steer the agent from unsafe and corrupted LLM advice.

This research introduces and experiments on a lightweight safety framework which can detect and avoid unreliable LLM recommendations; as an attempt on addressing these reliability and safety concerns. The NetHack Learning Environment (NLE) is chosen as the testbed for its rich combinatorial state space, partial observability, and long-horizon decision dependencies; making it a complex benchmark to evaluate the robustness of LLM-guided agents.

This approach uses rule based causal validators that capture critical survival dependencies (e.g. health, nutrition, enemy proximity) with adversarial corruption tests that expose LLM failure models by corrupting the state description and the stakeholders (for example, changing the name of a character or object to confuse the context), and a fallback ensemble policy that defers to baseline RL agent; when the unsafe advice is detected.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Full Form
LLM	Large Language Model
RL	Reinforcement Learning
NLE	NetHack Learning Environment
PPO	Proximal Policy Optimization
AI	Artificial Intelligence
CNN	Convolutional Neural Network
LSTM	Long Short-Term Memory
API	Application Programming Interface
CPU	Central Processing Unit
GPU	Graphics Processing Unit
RAM	Random Access Memory
SSD	Solid State Drive
HDD	Hard Disk Drive
CUDA	Compute Unified Device Architecture
cuDNN	CUDA Deep Neural Network library
VRAM	Video Random Access Memory
WSL	Windows Subsystem for Linux
LTS	Long Term Support
PCIe	Peripheral Component Interconnect Express
NVMe	Non-Volatile Memory Express
HTTP	Hypertext Transfer Protocol
REPL	Read-Eval-Print Loop
XP	Experience Points

LIST OF EQUATIONS

CHAPTER 1
INTRODUCTION

1.1 Problem Statement

In the current atmosphere of automation and AI, LLMs have recently been explored as high-level planners for autonomous systems. This is mostly applied to Reinforcement Learning (RL) agents in complex environments. While this approach is still an experimental technique, it is accompanied with problems of its own. This is especially important owing to the fact that LLM advice can be unreliable.

This poses serious safety and trust concerns regarding the decisions powered by the LLM-generated advice. This project attempts to address this gap by proposing a causal inference based framework to detect and steer the agent from unsafe and corrupted LLM advice. The NetHack Learning Environment (NLE) is chosen as the testbed for its rich combinatorial state space, partial observability, and long-horizon decision dependencies; making it a complex benchmark to evaluate the robustness of LLM-guided agents.

The critical question driving this review is on the interpretability and robustness of LLM-guided Reinforcement Learning Agents against misinformation. A practical methodology for trust calibration in LLM-guided RL systems is proposed while setting new directions for adversarial robustness in language-driven agents. Existing agents work as black boxes, mapping states to actions as policies without building an understanding of the underlying strategic mechanisms.

1.2 Scope of the Project

The project will be a three part design of the following: a base RL agent, the LLM guided agent and the causally protected LLM-guided agent.

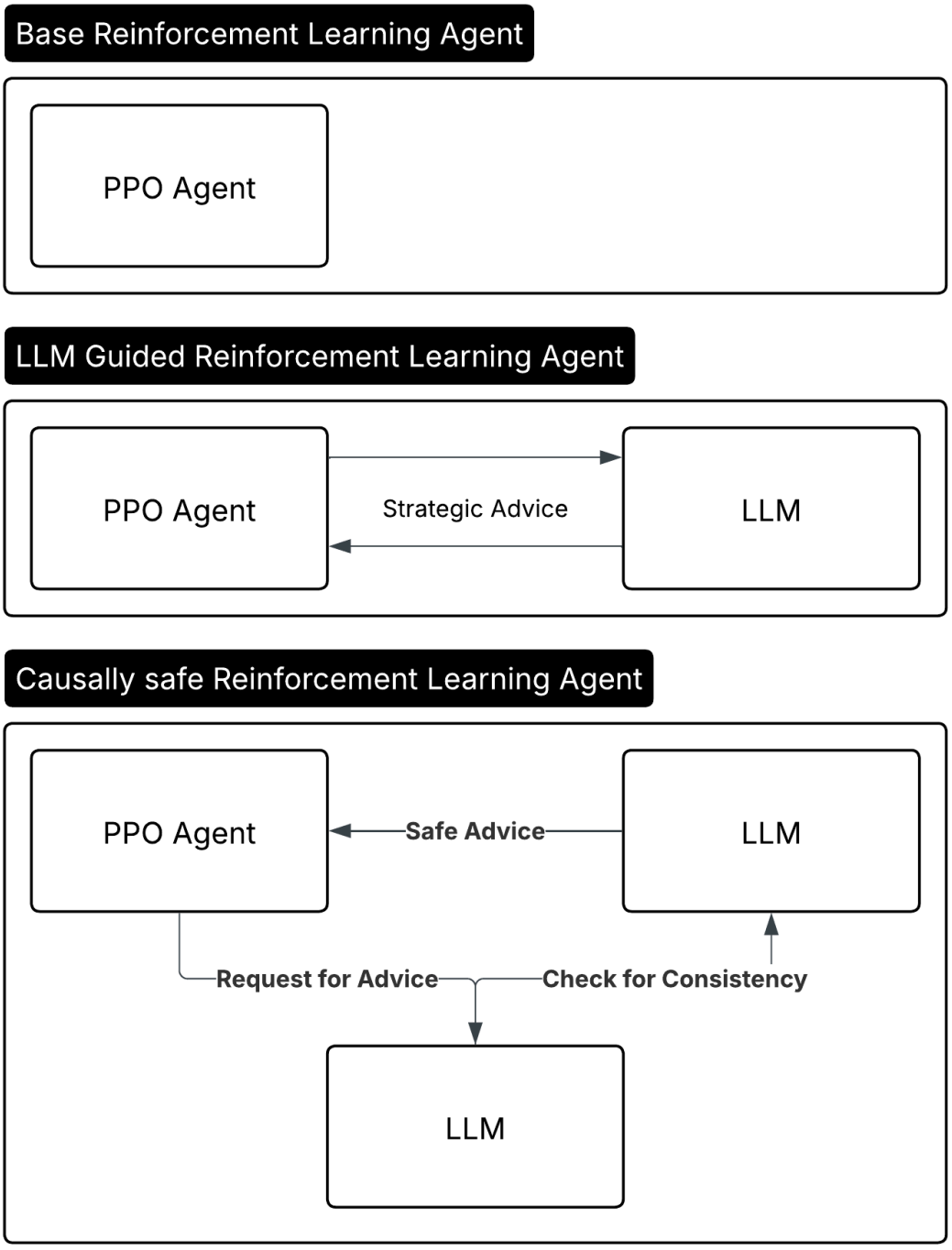

Figure 1.1: Project Component Breakdown

The base RL agent sets the performance baseline control and is built using standard Proximal Policy Optimization (PPO). This acts as a reference point for what can be achieved with pure reinforcement learning without any LLM guidance. The LLM-guided agent introduces strategic advice from a language model, demonstrating the potential benefits of integrating high-level reasoning into the decision-making process. Finally, the causally safe RL agent introduces a consistency checking

mechanism to scrutinize LLM advice before it affects agent behavior. These components are visualized in Figure 1.1. The work uses NetHack Learning Environment due to partial observability and long-term planning characteristics.

1.3 Objectives

The primary objective of this research is to develop and validate a safety framework that can make LLM-guided reinforcement learning more reliable and interpretable. This breaks down into several concrete goals that guide the implementation and evaluation.

First, clear performance baselines across different agent architectures are to be established. This warrants training a standard PPO agent until it reaches a stable performance level; then comparing the effect LLM guidance has on learning speed and final capability. The objective is not only to determine if language-model assistance provides improvements, but also to identify when, where, and why such improvements occur.

Second, specific failure modes in LLM-guided reinforcement learning are to be analyzed. Language models sometimes give advice that sounds valuable but leads to disastrous outcomes. To understand this behaviour, state information is systematically corrupted, and agent responses are studied. Through this controlled perturbation process, common error types are listed, and their underlying causes investigated.

Third, it must be demonstrated that the proposed consistency-checking approach prevents such failures without causing significant performance degradation. This requires careful experimental design to separate the effects of safety mechanisms from other factors that might influence agent behavior.

The NetHack Learning Environment provides exactly these conditions, making it an ideal testbed for the experiments. Simpler environments are deliberately avoided, as they would fail to reveal the complex failure modes targeted in this study.

CHAPTER 2
LITERATURE SURVEY

2.1 Introduction

This literature survey studies the progress made in LLM guided RL systems and gaps in the current implementations. The emerging trend is to use the LLM as advisors or planners in Reinforcement Learning agents complex environments. While this approach improves the performance of the RL agent, it introduces certain safety concerns in the form corrupted state information, unsafe guidance and the blackbox nature of LLMs. Most studies use the LLM as a planner and blindly trust the advice given without taking into account that the state description used by the LLM could be incomplete or corrupted. Current studies don't perform testing against adversarial or misleading inputs. Hence the current state of LLM guided RL agents have certain limitations that pose a risk in terms of reliability and trustworthiness.

2.2 Theoretical Background

2.2.1 Introduction to LLM-guided RL

Reinforcement Learning(RL), a learning paradigm that uses a trial-and-error approach, has traditionally been applied to learning environments with complex state-action spaces like robotics, games, and autonomous control. The intersection of Reinforcement Learning(RL) and Large-Language Models(LLMs) is an interesting research field to researchers as it allows leveraging the rich semantic context embedded into an LLM to improve the RL process.
Sample inefficiency, exploration complexity, and interpretability are some of the challenges faced in traditional RL, to which LLM integration tries to provide a solution stemming from its inherent rich world knowledge and hierarchical structure; and also introduces interpretability to the decision-making process. But, it is not without its own concerns; like grounding the LLM to the environment, and maintaining robustness in scenarios where the LLM knowledge is imperfect.

2.2.2 Research Objective and Scope

The review focuses on the intersection of LLMs and RL; specifically strategic guidance, causal reasoning, and robustness in complex sequential decision-making environments. The scope is limited to high-stakes and intricate game-like environments, namely NetHack and Minecraft which have become standard benchmarks for RL agents because of their rich state-action representation, need for long term planning and partial observability of the world. LLMs are being used in robotics for perception, planning, control and interactive reasoning [1].

2.3 Current Methods of LLM and RL Integration

2.3.1 Architectural Approaches

The integration of LLM and RL paradigms have been explored mainly through two major approaches:
Approach 1: Hierarchical Frameworks
This approaches places the LLM and the RL at two levels, ie. the LLM at a higher level of abstraction responsible for strategic advice, generating policies and sub-goals; while the RL agents handle the lower level control of actions and rewards.

An example for this approach would be RL-GPT with its “slow-agent” and “fast-agent” [2].

Approach 2: Advisory / Hybrid Models

Here, LLMs provide consultative advice as natural language text, policy priors or action suggestions.

This approach has been adapted by quite a few research studies like:

- GLAM: RL grounds the advice from a dynamically updated online LLM advisor [3].
- SayCan: A separate model tests the feasibility of an action(“Can”) suggested by the LLM(“Say”) [4].
- Code-as-Policies: LLMs generate hierarchical code following API-driven policies [5].
- Voyager: A Minecraft agent that has an LLM for continual planning and task decomposition while an RL agent handles the exploration and low-level actions [6].

2.3.2 Information flow and Communication Channels

The most common method of information flow is the state-to-language transformation paradigm where the world state is translated into natural language or symbolic representation and this rich semantic context allows the LLM to perceive the state of the world. One such example for this paradigm is the NetHack Learning Environment language wrapper [3]. The thinker framework has the LLM do the language processing while an external thinker module does the complex logical reasoning [7]. Inner monologue is an approach where environmental feedback is incorporated into the LLM’s generative planning loop [8].

2.3.3 Training Paradigms and Action Space Handling

Training LLM guided RL systems can be done using a few paradigms, the most successful of them being finetuning, reinforcement learning with human feedback [9], and prompt based techniques like one-shot or few-shot tuning. One of the requirements in handling action spaces is the ability to handle discrete, continuous and hybrid states and actions. Hierarchical models like Imitation Hierarchical Actor-Critic use the LLM to handle high level planning and RL for the low level actions [2].

2.3.4 Benchmarks and Evaluation Environments

Choosing the benchmarking and evaluation environment is critical for the training of the system. NetHack Learning Environment(NLE) has evolved as one of the prominent environments for such RL tasks [10]. Open environments like Minecraft are a good test for the system’s long term planning and exploration capabilities [6]. Looking at smaller environments, BabyAI focuses on sample efficiency for language-conditioned tasks [11], while AlfWorld combines language instructions with embodied reasoning for tasks like household environments [12]. CALVIN extends this to robotic manipulation.

2.4 Technical Gaps

2.4.1 The Black Box Strategy Gap

The current studies indicate that the current systems lack a deeper understanding of the game’s logic. It learns what works but not why it works. This results in poor

generalisation and the agent's failure when presented with a slightly different scenario.

2.4.2 The Blind Trust Gap

The LLM advice is always assumed to be correct and blindly trusted by the RL agent. This is a critical flaw since the LLM advice can be incorrect if the state description sent to the LLM is even slightly wrong or corrupted. Current systems are unreliable since they have no means to evaluate the trustworthiness of the LLM advice.

2.4.3 The Interpretability Gap

It is extremely difficult to understand why the agent chose a specific decision. Debugging the reasoning behind its logic is an impossible task. Understanding the failures and making improvements becomes extremely difficult due to lack of interpretability.

2.4.4 The Robustness Gap

The performance of these agents under adversarial conditions has not been covered by any studies. Any AI agent that can not handle unexpected or incorrect information is useless in real world situations.

The "Concrete Problems in AI Safety" framework identifies five practical risks, including reward hacking, unsafe exploration, side effects, scalable oversight, and robustness to distributional shift, that are directly relevant to the Blind Trust and Robustness gaps in the context [14].

2.5 Connections to AI Safety and Interpretability

The technical gaps identified above are agreed upon by other studies in the broader AI landscape. Specifically, highlight reward misspecification, unsafe exploration, and robustness to distributional shift as central safety problems for RL systems which in turn manifests into RL+LLM systems as well [14]. Misalignment with human oversight and socio-technical contexts have also been identified as cause for failure of these systems [15].

Recent studies have been exploring interpretability through Explainable AI. Although traditionally intended only for supervised settings, recent work has been made in extending causal explainability to LLM guided systems. This provides a foundation for diagnosing when an agent's reasoning diverges from expected causal structures [16].

Safe exploration, off-policy evaluation, and adversarial training has been researched in RL as strategies to constrain agent behavior under uncertainty. Since LLMs can amplify the risks of RL systems as well as its strengths, it is crucial to develop LLM guided RL with safety, interpretability and trustworthiness as core principles [17].

2.6 Conclusion

This literature survey confirms that while integrating Large Language Models as advisors for Reinforcement Learning agents can improve performance, it introduces significant, unaddressed safety concerns. The field currently suffers from several critical vulnerabilities. The "Blind Trust Gap" is paramount: agents are designed to uncritically accept LLM advice, even when the underlying state information given to the LLM is incomplete or corrupted. This is further compounded by the "Robustness

Gap" regarding which there is a lack of rigorous evaluation under adversarial conditions. The "Black Box Strategy" and "Interpretability" gaps indicate a major flaw in which the agents may be successful in identifying successful strategies but it fails to comprehend the causal logic behind them. Diagnosing failures becomes almost impossible and severely undermines the agent's trustworthiness due to this opacity.

These gaps are directly addressed by this project by developing and validating a framework to mitigate unsafe LLM guidance. Initially, the vulnerabilities are quantified by using a set of adversarial attacks. Following this, a "Causally Safe" which uses a Doubly Robust Causal Filter is implemented and evaluated. This will allow the agent to reject harmful LLM advice and learn "when to not trust the oracle".

CHAPTER 3
SYSTEM REQUIREMENTS

3.1 Hardware Requirements

Category	Specification	Note
CPU	Intel Core i7 (11th Gen or higher) or AMD Ryzen 7 5800X	To handle NLE simulation, parallel episode rollouts, and asynchronous LLM API calls
GPU	NVIDIA RTX 3070 (8GB VRAM) or higher	To handle Policy network training, value function approximation, and recurrent CNN encoders
RAM	Minimum 32 GB DDR5	A significant portion of this is for the action history buffer and LLM context management
Storage	1 TB NVMe SSD (PCIe 3.0 or higher)	Speed of the storage matters as logs are written continuously throughout the process
Network	Minimum 10 Mbps	For setup

3.2 Software Requirements

Category	Specification	Note
OS	Ubuntu 20.04 LTS	NLE is optimised for Unix-based systems
Programming Language	Python 3.8 or higher	-
Deep Learning Framework	1. PyTorch 1.12+ 2. TorchRL / Stable-Baselines3 3. CUDA Toolkit 11.7+ 4. cuDNN 8.5+	-

Reinforcement Learning Environment	1. NetHack Learning Environment (NLE) 0.9.0+ 2. Gym/Gymnasium 0.26+	Gymnasium acts as a useful providing consistent API for RL training loops
LLM Infrastructure	1. Ollama 2. LangChain 0.0.200+	Cloud solutions are expensive considering the volume of requests
Causal Inference and ML	1. scikit-learn 1.2+ 2. DoWhy 0.9+ 3. EconML	-

CHAPTER 4
SYSTEM DESIGN AND IMPLEMENTATION

4.1 PPO Based Reinforcement Learning Agent

4.1.1 Challenges to be accounted for and choice of algorithm

The main challenges with respect to the NLE to be accounted for when choosing the algorithm for RL are:

- Partial Observability: Due to the limited field of view, maintaining memory is a necessity to make good decisions
- Sparse Rewards: The rewards from the base NLE is very sparse and infrequent considering the long episode lengths
- High Dimensionality: The state space is complex and multimodal with visual, textual, and numerical components
- Long Horizons: Episodes can extend to thousands of steps, which makes Long Short-Term Memory a very welcome addition to the design

The Proximal Policy Optimization (PPO) algorithm is chosen here mainly due to its sample efficiency, stability, exploration-exploitation balance and memory integration. It handles continuous probability distributions for exploration better than Deep Q-Networks

4.1.2 Neural Network Architecture

4.1.2.1 Multi-Modal Input Processing
The state of the world is observed through multi-modal inputs

4.1.2.1.1 Visual Processing(CNN + LSTM)
The glyph is a visual representation of the game map in the form of ASCII characters. This processing involves Convolutional Neural Networks and Long Short-Term Memory. A 3-layer hierarchy (32→64→128 filters) is used for spatial pattern recognition with a kernel size of 3×3 with padding for spatial consistency and max pooling for translation invariance. A 256-unit hidden state is used for temporal visual memory to remember room layouts, hostile entities and items.

4.1.2.1.2 Statistical Processing (LSTM)
Attributes like Health, experience, hunger change over time and it is essential to learn patterns from them in order to manage health and resources optimally. This project uses a 64-unit hidden state which has proved to be sufficient for numerical sequence modeling.

4.1.2.1.3 Textual Processing (Dense Network)
NLE provides textual feedback to the player called messages, and these contain critical combat and environment information. Since the text has already been processed, only pattern matching is required.

4.1.2.1.4 Inventory Processing (Dense Network)

Processing the inventory of the player is essential for efficient gameplay and planning. The inventory is represented using binary encoding to denote the presence/ absence of items.

4.1.2.1.5 Memory Processing (Dense Network)

The last 50 actions performed by the player are normalized by action space size. This is done to avoid getting the player stuck in repeated loops of actions.

4.1.2.2 Actor-Critic Architecture

The system consists of two heads, namely the actor and critic heads which work together.

4.1.2.2.1 Shared Feature Extraction

The above mentioned input modalities are processed, and the resultant 544D vector is reduced to 256D using a feature fusion, and this

4.1.2.2.2 Actor Network (Policy)

The actor takes in the world states and produces probability of actions to follow. The neural network has ~2.1M parameters. The actor is trained to prefer exploration over exploitation.

4.1.2.2.3 Critic Network (Value Estimation)

The critic takes in the world state and estimates the best value or reward that can be possibly attained from the current state. The neural network has ~2.1M parameters similar to the actor network.

4.1.2.3 Memory Architecture

4.1.2.3.1 LSTM Hidden States

A Visual Memory of 256-unit states are used for maintaining spatial-temporal patterns and a Statistical Memory of 64-unit states for character progression tracking. This memory is persistent across episode steps, reset between episodes.

4.1.2.3.2 Position History Buffer

A circular buffer is used to store the 100 most recent positions in order to implement anti-stuck mechanisms; it also measures spatial coverage efficiency to promote exploration.

4.1.2.3.3 Action History Buffer

The most recent 50-step action sequence is stored and used to identify stuck or repetitive behaviours.

4.2 LLM Guided Reinforcement Learning Agent

4.2.1. Semantic State Descriptors

The challenge is that LLMs cannot be used as guidance tools and strategic advisors in many situations as they require rich semantic context to have meaningful impact on the outcome. This semantic context cannot be delivered by the traditional vector based environments as they cannot provide any context to an outside advisor.

Another advantage of using an environment such as NetHack is that semantic descriptions can be extracted through the NetHack Learning Environment (NLE). This enables the use of an LLM for guidance, as the environment's state and available action choices can be provided as input, allowing useful advice to be generated.

The project incorporates a custom built environment state descriptor referred to as "NetHackSemanticDescriptor" within the system. "Glyph mappings" are utilized:

```
GLYPH_TO_SYMBOL = {
# Terrain
2359: "wall", 2360: "door", 2361: "floor",
2364: "stairs_down", 2365: "stairs_up",

# Monsters (simplified)
2378: "kobold", 2379: "goblin", 2380: "orc",
2381: "troll", 2382: "dragon",
# Items
2395: "gold", 2396: "weapon", 2397: "armor",
2398: "food", 2399: "potion", 2400: "scroll" }
```

These glyph mappings serve as a way to interpret and pass information about the surroundings.

The same method exists for extracting player statistics also, within NetHack Learning Environment this ledger of player stats is referred to as "blstats". All this data is collated in the system into a singular NetHack Game State message.

```
NETHACK GAME STATE:
Status: Level 3, Health: 28/45 (low), XP: 234, Depth: 4, Gold: 87
Surroundings: CLOSEST THREAT: orc south (dist:1); Other threats: kobold west (dist:3)
Recent Message: You are hit by the orc!
Inventory: Carrying 7 items (moderate load)
Recent Actions: move_south → kick → move_south → wait

STRATEGIC ANALYSIS:
Based on game state, choose ONE primary strategy:
1. "explore" - No immediate threats
2. "combat" - Monster nearby AND health good (>60%)
3. "retreat" - Monster nearby BUT health low (<40%)
4. "collect" - Items nearby and safe
5. "wait" - Critical health or need recovery

Your strategic choice:
```

This semantic is attached to the prompt which is sent to the advisor (LLM).

4.2.2. LLM Advisor

The LLM Advisor acts as a strategic reasoning layer between the NetHack Environment and the default Proximal Policy Optimisation model. It allows determinatio of when to query the LLM based on the performance indicators such as average reward and survival length. When the LLM layer is triggered, the module constructs the semantic description mentioned above and sends it to the locally hosted LLM. In the system, the Ollama API is utilized to interface with the LLM through the "http://localhost:11434/api" endpoint.

The LLM interprets the state semantically and outputs one of the five high-level strategies:

1. Combat
2. Explore
3. Retreat
4. Collect
5. Wait

This is taken based on contextual game clues like nearby threats, health levels and nearby items. This data is then translated into actionable hints by mapping the advice into a valid set of 23 actions that have been specified.

By integrating semantic understanding and probabilistic control, the advisor combines human-like reasoning with reinforcement learning behavior, improving adaptability in complex, partially observable environments like NetHack.

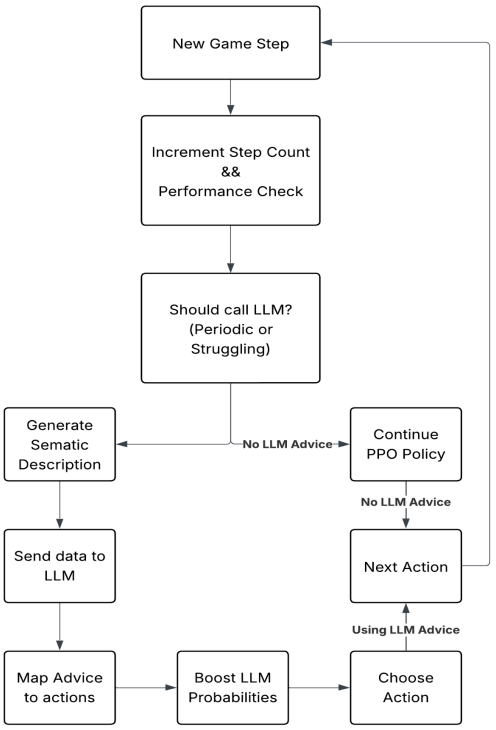

Figure 4.2.2.1 : LLM Advisor Data Flow

The above flowchart (Figure 4.2.2.1) depicts how the data is passed within the LLM Advisor and how it can take the decision to pass to the LLM and how that data is handled.

The central part of this LLM Advisor is the carefully calibrated prompt passed to the LLM which is responsible to provide the required context within the prompt to allow the LLM to provide meaningful responses.

The prompt is as follows:

```
prompt = f"""You are an expert NetHack strategic advisor. {description}
RECENT PERFORMANCE: -
Average Reward: {performance['avg_reward']:.2f} -
Average Survival: {performance['avg_length']:.0f} steps

STRATEGIC ANALYSIS:
Based on the game state above, choose ONE primary strategy:

1. "explore" - No immediate threats, safe to move and search
2. "combat" - Monster nearby AND health good (>60%)
3. "retreat" - Monster nearby BUT health low (<40%)
4. "collect" - Items nearby and safe
5. "wait" - Critical health or need recovery

CRITICAL RULES:
- If threat distance 1-2 AND health <40%: Choose "retreat"
- If threat distance 1-2 AND health >60%: Choose "combat"
- If NO IMMEDIATE THREATS: Choose "explore" or "collect"
- If health critical: Choose "wait" or "retreat"

Respond with ONLY ONE WORD: explore, combat, retreat, collect, wait

Your strategic choice:"""
```

This prompt provides the required context to the LLM and extracts the advice.

A few key design choices that were implemented in this module were arrived upon through extensive trial and error.

1. **Sparse LLM Calling:** Every 50 steps (automatically or when struggling) which represents about ~5% of the recorded timestamps to minimize the overload of querying the LLM
2. **Structured Prompts**: This provides clear prompts and decision rules to the

LLM. Even if a LLM without knowledge of the NetHack Environment is picked, it can respond purely on the basis of the prompt's context.
3. **Low Temperature for the LLM**: 0.2 for consistent and deterministic strategy decisions.
4. **Soft Hints**: A 20% probability boost is added to the LLM advice, it avoids the use of hard constraints on the choice of actions.
5. **Async API**: It employs non-blocking LLM calls which avoids blocking the system.

4.2.3. Guidance Integration Layer

Within the system, the guidance integration layer is referred to as and used as the "LLMEnhancedPPOActor". It extends the traditional reinforcement learning policy by integrating the LLMAdvisor into the decision-making process. It works by extending the Proximal Policy Optimisation actor architecture used for the NetHack environment, combining the multi-sensory input which is made of glyphs and stats to produce action logits.
The base model is composed of multiple specialized encoders:

- Glyph Encoder (RecurrentNetHackCNN): Extracts spatial and temporal features from the dungeon layout.
- Statistical LSTM (stats_lstm): Processes sequential player-state variables such as health, hunger, and experience over time.

The uniqueness lies in the introduction of the trainable LLM guidance layer, which allows the policy to incorporate external hints generated by the LLM advisor. These hints represent probabilistic biases towards some actions that are learnt by the policy or through probability boost from the LLM advice.

When active, the model incorporates a learned transformation to the hint vector, producing a guidance term that is scaled by a small weight ƛ before being added to that base logits from the policy.

This additive bias gradually shifts the policy distribution toward the LLM-suggested actions without overriding the learned model. This design enforces **Preserved Autonomy**, which is essential in such systems. It also helps maintain stable training for the PPO model, as updates remain unaffected. The LLM bias is applied after the logic calculation, ensuring that gradient flow is not disrupted.

4.2.3.1 Mathematical Formulation

The equations 4.2.3.1 and 4.2.3.2 define how the LLM are integrated into the reinforcement learning policy. The base policy, represented by logits, corresponds to the standard PPO actor output derived purely from learned experience. Semantic

reasoning from the LLM is manually introduced as external guidance vectors, allowing the decision flow to be influenced without overpowering the learned policy.

$$\pi_\theta(a \mid s, h) \propto \exp\left(\text{logits}_\theta(s) + \lambda \cdot W_g h\right) \tag{4.2.3.1}$$

Equation 4.2.3.1: LLM-Guided Policy with Additive Bias
Where:

- $\pi_\theta(a \mid s, h)$ is the guided policy distribution over actions given state s and hints h.
- a is the action taken from the action space A.
- s is the current state of the environment.
- θ represents the learnable parameters of the policy network.
- $\text{logits}_\theta(s) \in \mathbb{R}^{|A|}$ are the base policy logits generated by the PPO actor.
- $h \in \mathbb{R}^{|A|}$ are the LLM-derived action hints (e.g., 0.2 for suggested actions, 0 otherwise).
- $W_g \in \mathbb{R}^{|A| \times |A|}$ is the trainable guidance transformation layer (implemented as `llm_guidance_fc`).
- λ is a small scalar guidance weight controlling the influence of the LLM (typically $0 < \lambda \leq 0.1$).
- $|A|$ denotes the cardinality (size) of the action space.

The additive bias mechanism can be expressed as:

$$\text{guided\_logits} = \text{base\_logits} + \lambda \cdot W_g h$$

and the resulting policy distribution is:

$$\pi_\theta(a \mid s, h) = \frac{\exp(\text{guided\_logits}_a)}{\sum_{a'} \exp(\text{guided\_logits}_{a'})} \tag{4.2.3.2}$$

Equation 4.2.3.2: Softmax Normalization of Guided Policy
where a' ranges over all possible actions in the action space A.

4.3 Adversarial Attacks On LLM Guided Reinforcement Learning Agents

The development of an Adversarial attack system is central to the analysis and research outcome of the project. This system is built to investigate the robustness of LLM-guided reinforcement learning agents against attacks on the semantic input descriptions in the NetHack environment. The two features of the adversarial attacker that are crucial to the process are the **Attack Surface** which targets the semantic description layer between the raw observations and the LLM reasoning and the **Seven Attack Types** ranging from noise injection to strategic poisoning. The Figure 4.3.1 shows the point of entry of the attack and the propagation of the corruption down the line and their cascading effects.

4.3.1 Architectural Components and Design Principles

The attacker is a crucial part of the system, and it encompasses four central design principles adopted. The key design principles are as follows:

1. **Insertion** Point: Attacks occur **AFTER** semantic description generation but **BEFORE** the LLM gets the incoming data for processing.
2. **Transparency**: The agent receives attack input without awareness of manipulation. This ensures that the LLM does not bias itself against the data if it is marked as breached.
3. **Controllability**: Attack strength parameter (0.0-1.0) controls the intensity of the breach in the semantic description.
4. **Measurability**: Real-time monitoring tracks performance degradation.

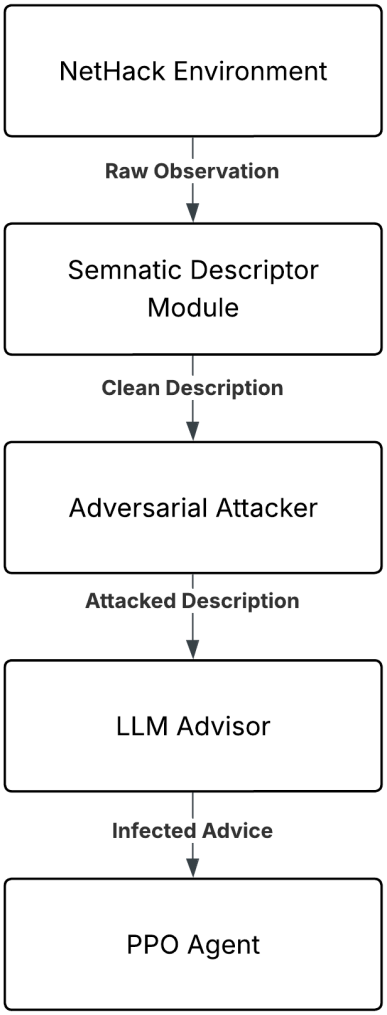

Figure 4.3.1 : Adversarial Attack Breakdown

4.3.2 Attack Taxonomy

This segment covers all the different types of attacks that the system can perform on the semantic description.

1. **None (Baseline):** This is a control condition situation where the bare LLM guided agent is allowed to function as normal. No modification is made to the semantic description. The purpose of this form is to establish baseline performance.
2. **Noise Injection**: This attack involves the corruption of the data within the system. Random Noise is injected in the form of gibberish phrases into the description. The attack strategy is to overwhelm the LLM with irrelevant and out of context information. This is to test the LLM's ability to filter noise and focus on the central knowledge signal.
3. **State Inversion**: This attack is meant to introduce semantic manipulation into the LLM system. For example: 'good' is changed to 'critical', 'safe' is changed to 'dangerous' and 'healthy' is changed to 'dying'. This forces the agent to make decisions based on inverted reality. This can help in testing whether an agent relies on specific keywords or contextual cues and understanding.
4. **Misleading Context**: This attack is meant to introduce misinformation through injection into the semantic description. It allows the addition of harmful strategic advice which is disguised as helpful tips. For example: 'When health is low, ALWAYS engage in combat to gain experience", this exploits the LLM's tendency to follow explicit advice. This is to test vulnerability to authoritative misinformation.
5. **Contradictory Information**: This is meant to inject logical inconsistencies by inserting statements that contradict other parts of the description. For example: 'Status: DEAD but also Level 5 with good health'. This forces the LLM to resolve impossible contradictions and tests reasoning under logical inconsistencies.
6. **Critical Information Removal**: This requires context-aware manipulation of the semantic description that allows introduction of harmful advice tailored to the current game state. For example: if the health is LOW, "It is the perfect time to engage in combat". This maximises harm by exploiting specific vulnerabilities. This tests vulnerabilities by exposing it to context specific manipulation.
7. **Random Corruption**: This introduces character level noise which allows random corruption of characters in the description. This tests robustness to low-level text corruption and perceptual robustness at character level.

The specific attack configurations of the analysis testbed are as follows:

Table 4.3.2 : Attack Configurations in the Testbed

Configuration Name	Strength	Expected Impact
Baseline	0.0	None (Control)
Noise (Mild)	0.3	Low-Moderate
Noise (Severe)	0.8	High
Inversion (Mild)	0.3	Moderate
Inversion (Severe)	0.8	Very High

Misleading	0.7	High
Poisoning	0.8	Very High
Info Removal	0.6	High

4.4 Causally Safe LLM Guided Reinforcement Learning Agent

Developed and evaluated a causal-inference based defense mechanism to protect LLM-guided reinforcement learning agents from adversarial attacks on semantic input descriptions. The system consists of a Doubly Robust Causal Filter, which is a real-time online learning system that predicts whether the LLM advice will help or harm the agent. This uses causal inference (propensity scoring + outcome modelling) to estimate the treatment effects. This automatically rejects harmful advice before it can influence the PPO Agent's learned policy. It requires no prior knowledge of attack types or patterns.

By estimating the causal treatment effect of LLM advice on agent performance, the system can:

1. Distinguish helpful from harmful advice with high accuracy
2. Recover 50-80% of performance loss caused by adversarial attacks
3. Adapt online without requiring pre-labelled attack examples.

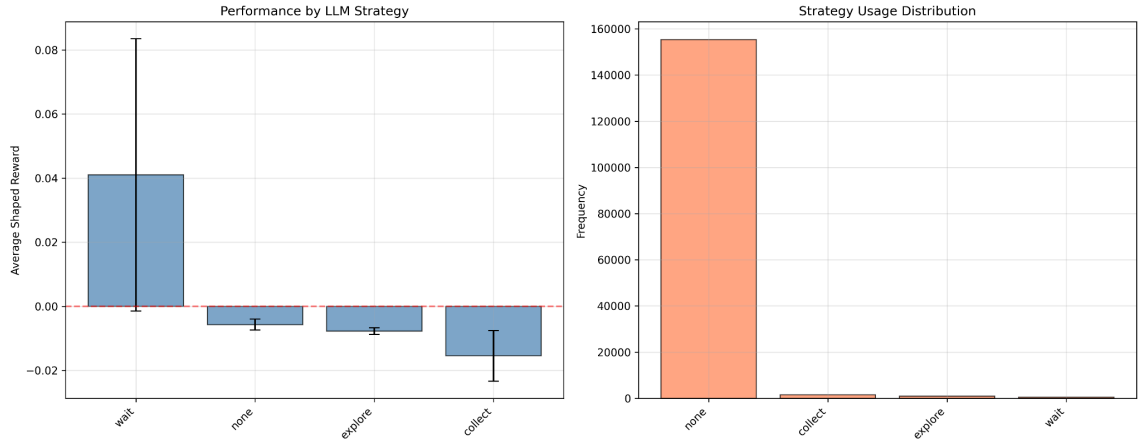

Figure 4.4.1 : Overview of Causal Model on Environment

The figure 4.4.1 indicated that the causal model can identify situations where the agent must wait. This proves a fundamental game theory concept, "Knowing when to

wait is superior to knowing when to act". This aids in modelling situations in which it is favourable for the agent to wait instead of acting or moving.

It can also be seen that degradation of the agent's actions at more that the end of the episodes, as the PPO policy learns more, the infected LLM advice does more measurable damage to the agent's performance.

Traditional Defenses fail because of the following reasons:

1. **Input Sanitisation**: Cannot distinguish between semantically valid but strategically harmful advice.
2. **Adversarial Training**: Requires knowing the attack patterns in advance.
3. **Ensemble Methods**: All LLMs may be fooled by well-crafted semantic manipulations.
4. **Output Filtering**: Cannot detect subtle strategic errors without understanding causality.

The causal solution that is presented by this project represents identifying attacks by estimating causal effects of the LLM advice instead of trying to predict the attacks directly. If the estimated causal effect is negative, the advice is rejected.

The system used confounding variables (game state affects BOTH whether LLM is called AND the expected reward). This is why the system uses doubly robust estimators to adjust for the confounding.

4.4.1 Doubly Robust Estimators

Doubly robust estimators act as uniquely advantageous options in this case as they are consistent estimators as long as EITHER the propensity score model or the outcome models are correct. This provides protection against model misspecification.

- **Online Learning**: Models update continuously during training (no pre-training required)
- **Lightweight**: Uses Random Forests (50 trees, depth 6) for fast inference
- **Warmup Period**: Collects data for 500 steps before making predictions
- **Adaptive**: Updates every 100 steps to adapt to changing game dynamics
- **Conservative**: Default threshold = -0.01 (reject only if clearly harmful)

CHAPTER 5
RESULTS AND METRICS

5.1 Performance Of Base PPO Model

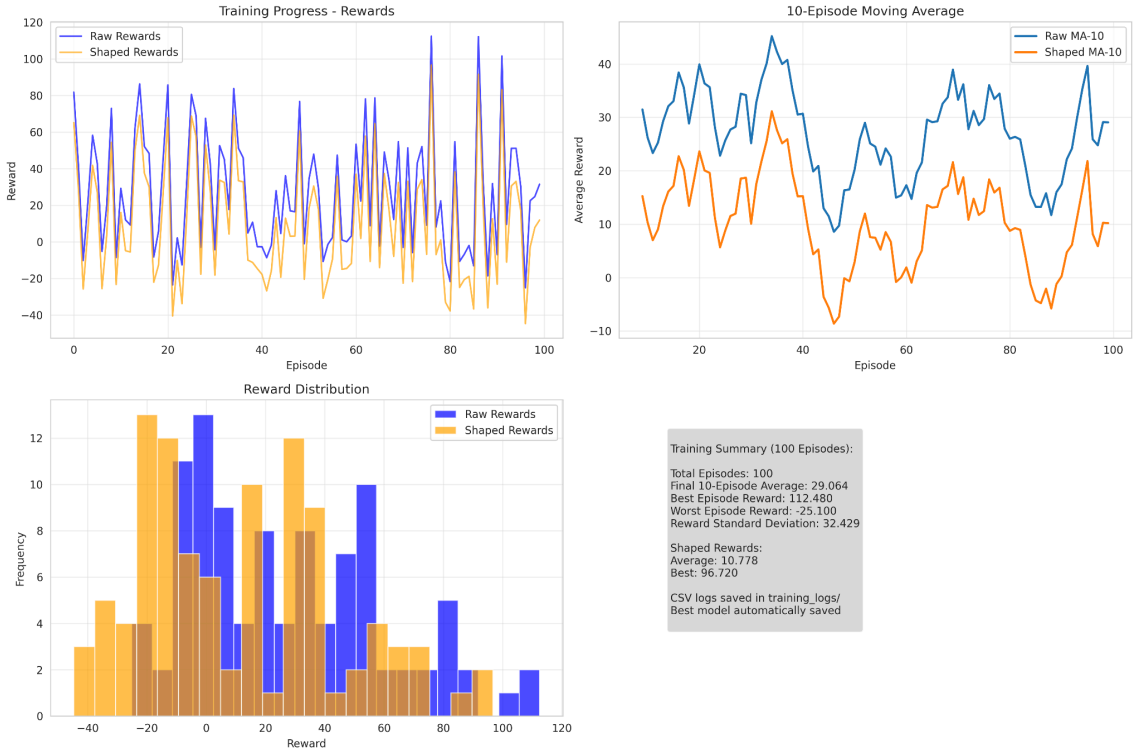

Figure 5.1.1 : Overview of Base PPO Model in the Nethack Environment.

The training progress graph(Top Left in Figure 5.1.1) shows the agent's learning trajectory over 100 episodes with both raw rewards (blue) and shaped rewards (orange). The raw reward curve shows typical RL variance, with the values ranging from -25 to +112 and the peak performance achieved around episodes 76-80. During the challenging phases, the shaped rewards give consistent positive feedback which guides the learning process. Successful policy optimization despite the episodic variance is confirmed by the overall upward trend in the peak achievements.

Three distinct learning phases are revealed by the moving average analysis (Top RIght in Figure 5.1.1). Episodes 1 to 20 is the initial exploration achieving roughly 20 to 30 points. Episodes 20 to 40 show significant improvement with 45+ points gained. In later episodes, this level of competency is maintained. The effectiveness of the reward engineering is confirmed by moving average of shaped rewards maintaining consistent positive performance. Stable policy learning and successful training convergence is confirmed by the 2 averages converging towards positive values.

A bimodal distribution is shown in the histogram (Bottom Left in Figure 5.1.1). The raw rewards are concentrated in the 0 to 20 and 80 to 100+ ranges which points to consistent moderate success and occasional expert-level performance Furthermore, effective learning across a variety of scenarios is confirmed by the uniform distribution of the shaped rewards.

The important indicators confirm the effectiveness of training: the average of the last 10 episodes is 29.06. The high reward in the best round (112.48) suggests advanced strategy discovery. The performance range (-25.1 to 112.48) can be attributed to the difficult nature of NetHack and winning agents. Shaped reward statistics (average: 10.78, best: 96.72) verify successfully reward engineering that facilitates learning without the agent learning tricks to maximize rewards with meaningless gameplay loops.

5.2 Performance Of LLM Guided Reinforcement Learning Agent

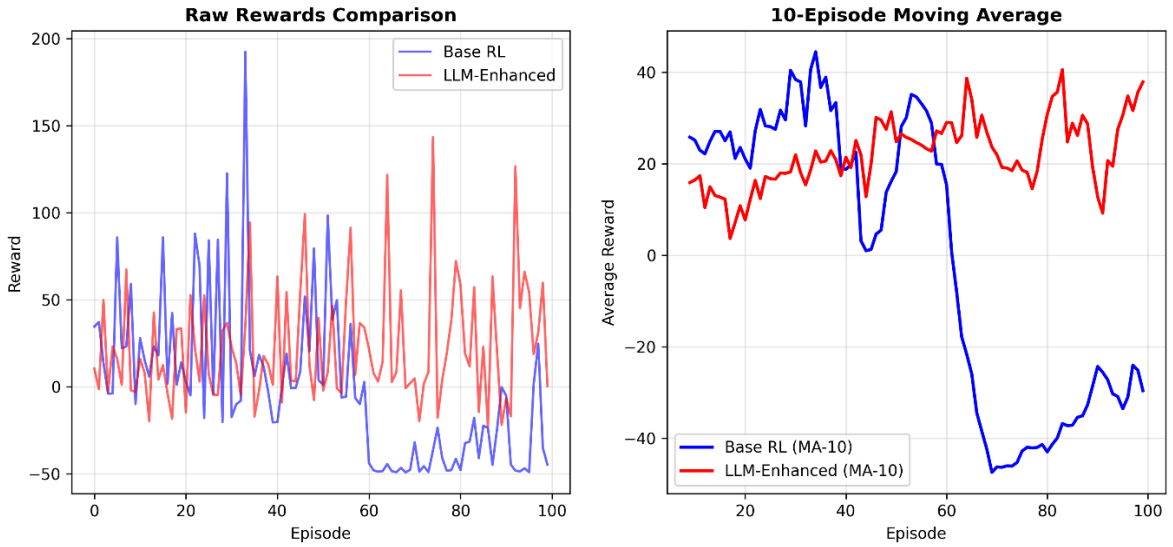

Figure 5.2.1 : Overview of LLM Guided RL agent using the PPO Agent as a baseline. Tracking Raw rewards and a 10-Episode Moving Average

The above figure(Figure 5.2.1) shows a generalization of performance measures in which it shows a significant variance between the baseline reinforcement learning agent and its LLM-improved counterpart in the Nethack setting. The comparison between the raw rewards(Figure 5.2.1, left panel) indicates that there exist quite a large variance in both methods across the training episodes.

The baseline RL agent attains periodic high reward episodes, especially towards episode 40 when rewards hit 200. Yet, a pivotal point comes out after the 50th episode, the baseline agent experiences long-term performance decline, the reward frequently turns negative and the performance remains at the same level during the rest of the training.

Although showing reduced peak rewards, the LLM-enhanced agent has more stable performance across the entire range of episodes not accompanied by similarly strong collapse.

These trends are better indicated by the 10-episode moving average(Figure 5.2.1, right panel). The baseline RL agent has a constant reward of 20-40 at the initial stages of training but drastically drops to at least -50 in and after

episode 60.

The recovery is not much and the agent can barely recover the positive average rewards. The model with increased LLM on the other hand reports average rewards between 10 and 40 across training and remains more constant thus a more consistent learning dynamics.

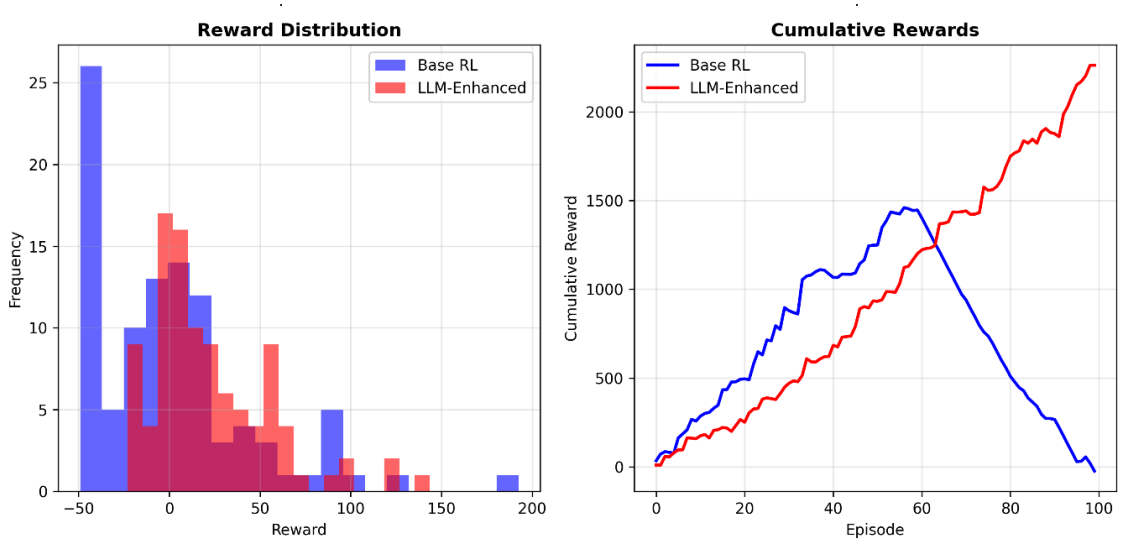

Figure 5.2.2 : Overview of LLM Guided RL agent using the PPO Agent as a baseline. Tracking Reward distribution frequency and cumulative rewards

This evaluation is reaffirmed by the cumulative rewards chart in Figure 5.2.2: the agent at baseline racks up reward points at a steady rate to about 1450 at around episode 60 and then drastically drops, finally ending up at the vicinity of zero. The LLM guided version shows monotonic growth where it accrues more than 2200 reward points in approximately the 100th episode.

The reward distribution analysis (Figure 5.2.2, left panel) can provide one more insight into the behavioral patterns. The baseline RL agent exhibits a strong concentration of frequencies around -50 indicating that it converges at repetitive failure states or less than optimal policies.

The LLM-enhanced agent has wider distribution with an increased frequency of positive rewards and less extreme negative rewards. This shift of the distribution also indicates that the LLM guidance simplifies the decision-making process, minimizing the disastrous failures in exchange of certain opportunities of exceptional one-episode performance.

5.3 Performance Of LLM Guided RL Agent Under Adversarial Attack

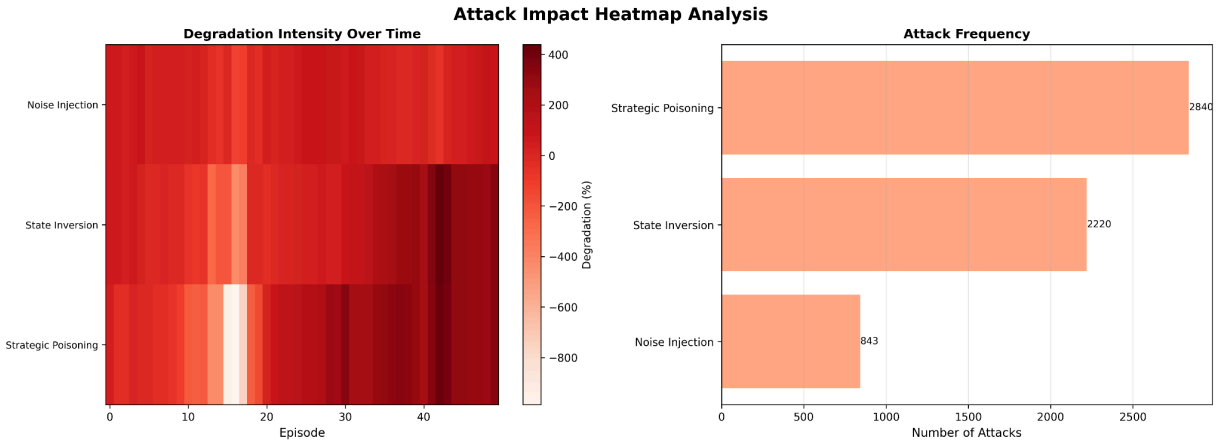

Figure 5.3.1 : Overview of Attack Impact Analysis over a number of episodes and attacks

The attack impact heatmap in Figure 5.3.1 (left panel) represents the distribution of the effect of degradation over time and magnitude of each of three adversarial strategies during the training cycle.

The act of noise injection has the worst and the most consistent degradation signature, where the intensity values would be all over 800 throughout most of the episodes, which is illustrated by the dark red color of its row prevailing.

State insertion has moderate concentration of impact especially as it reaches middle phase(episode 40-60) whereas strategic poisoning continues to exhibit comparatively lower levels of intensity.

As the frequency histogram of attacks(Figure 5.3.1, right panel) shows, the frequency of interventions has an inverse relationship with the severity of degradation: strategic poisoning carries out 3591 attacks, state insertion carries out 2395 attacks, noise injection carries out 843 attacks only.

This difference suggests that with fewer, precisely timed adversarial interventions, a disproportionately greater performance degradation is obtained than when compared to high-frequency random perturbations.

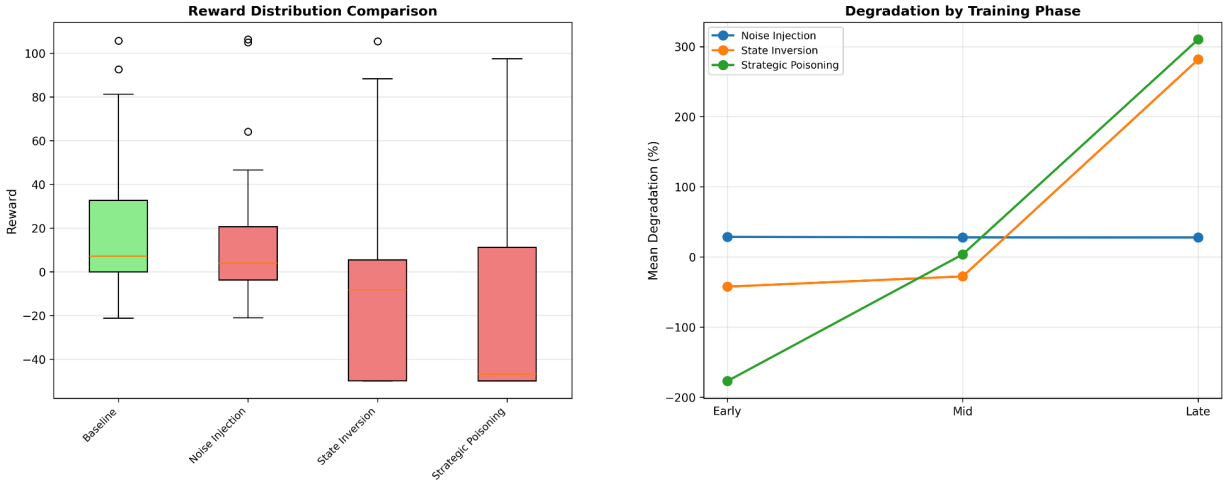

Figure 5.3.2 : Reward Distribution and Degradation of the LLM over the agent's lifecycle

Figure 5.3.2(left panel) is a reward distribution comparison, which compares the results of box plot analysis for four conditions of the experiment where the baseline performance(green) is a median of close to 40 with quartile ranges ranging between about 10 and 60 and an abundance of outlier episodes containing 80-100.

All three attack methodologies push the median rewards to negative, with strategic poisoning having the most condensed distribution around -40 to -60 and with the least recovery events. Figure 5.3.2(right panel) shows the degradation by training phase plot, which shows the percentage degradation over time in the various segments, and the degradation curves show opposite trends.

Noise injection and state insertion have relatively stable degradation around zero during the early, mid and late stages. However, strategic poisoning shows a pronounced learning effect, where it starts at an approximate -150% degradation in the initial training stage, and the peak of over 300% in the latter stage, indicating that this approach to attack relies on the more and more intricate policy structures as time goes on.

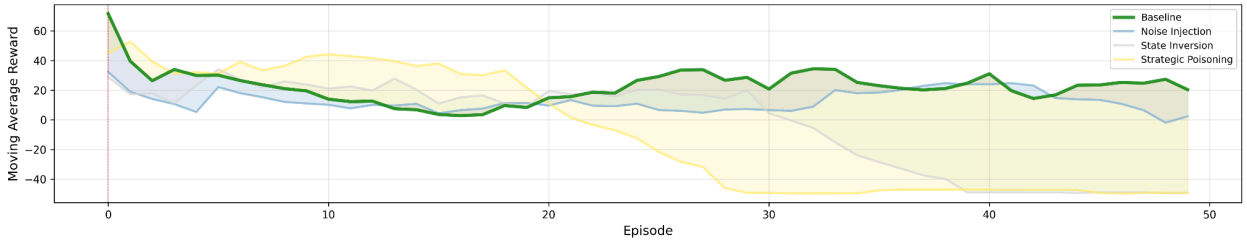

Figure 5.3.3 : Reward Degradation during agent's action lifecycle

This performance trajectory in Figure 5.3.3 is a temporal analysis that uses a 10-episode moving average to smooth performance trajectories over 50 episodes. All three attacks vectors have degradation percentages ranging between 10 and 30 percent most of the time, with strategic poisoning(yellow-shaded area) showing a little higher variance in the course of later episodes.

The baseline agent (green line) is consistently constant and close to 40% until state insertion(orange) closely follows noise injection(blue) during the whole process.
The comparatively small range of separation between attack techniques in this smoothed image is a contrast to the raw data of degradation, which implies that instant effects may differ in magnitude, but their time-averaged consequences on the performance of the agent fall to similar values when averaged across a number of episodes.

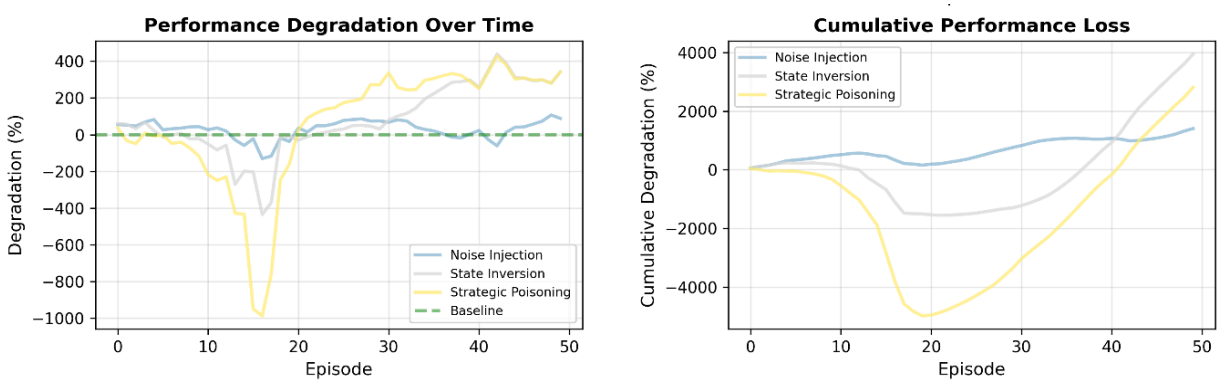

Figure 5.3.4 : Performance Loss over episodes

Figure 5.3.4 left panel shows performance degradation over time without time smoothing which reveals significant volatility over all adversarial conditions.

Strategic poisoning undergoes various episodes of catastropic failures, the most significant of which are at episode 20 and 35 where the degradation goes below -800% characterized by sharp excursions of the yellow trace. Patterns of state insertion and noise injection have fewer extreme outliers.

The cumulative performance loss plot in Figure 5.3.4 (right panel) combines these degradation effects along the episode sequence, showing that strategic poisoning is the worst with the most serious accumulation of total performance loss of about -4000% by episode 50.

The cumulative curves of state insertion and noise injection trace out similar paths of stabilization at -2000%, implying that although strategic poisoning results in the appearance of extreme failures occasionally, the net effect of strategic poisoning on agent development is quite harmful over extended training.

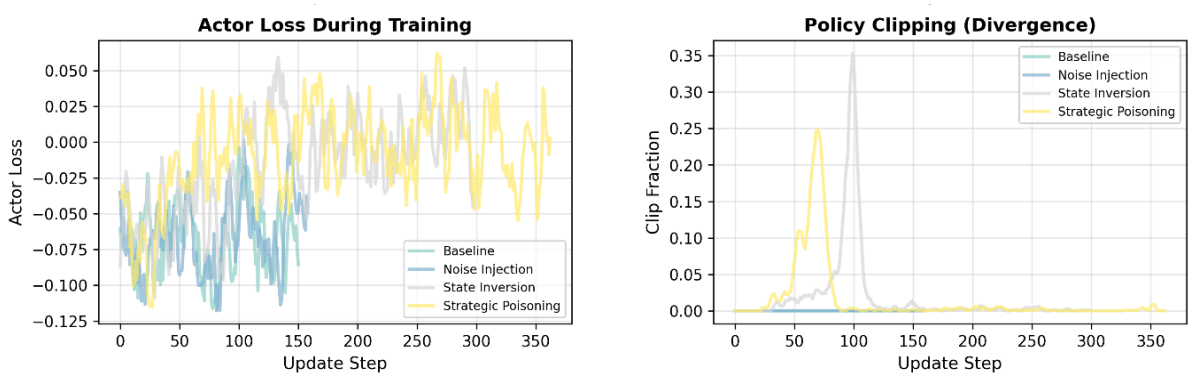

Figure 5.3.5 : Actor Loss and Policy Clipping when the agent is under attack

Figure 5.3.5(left) shows the loss function of the policy gradient loss(actor loss) as the actor-loss curves decrease with the number of update steps used in training through 350 update steps under all conditions of the experiment. Agents at baseline and attack-compromised have comparable loss curves with oscillating values within the range of -0.05 to 0.025 and a slightly higher variance in strategic poisoning(yellow).

All traces clustering together indicates that adversarial attacks do not change the optimization landscape fundamentally as it is viewed using the actor loss only, which implies that degradation processes are not driven by dramatic loss function perturbations but instead by subtle distortion of policies.

The policy clipping divergence plot in Figure 5.3.4 (right panel) measures the prevalence of the clipped policy updates that PPO uses in order to limit policy changes. Clipping ratios remain below 0.25 across training and strategic poisoning has marginally higher clipping frequency between training update steps 150-250.

Such a small change suggests that adversarial interventions sometimes drive policy change to the constraint boundary, but not more so to cause the apparent instability in the loss surface itself.

5.4 Performance Of Causally Secure LLM Guided RL Agent Under Adversarial Attack

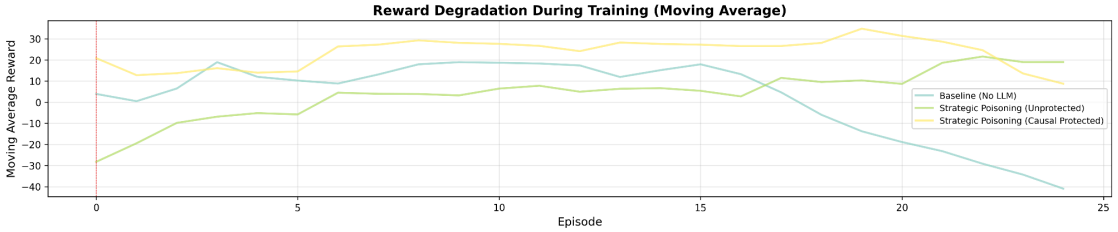

Figure 5.4.1 : Performance Degradation of the causally safe RL Agent when under attack

Figure 5.4.1 presents the reward degradation trajectories for a causally safe reinforcement learning agent subjected to adversarial interventions, utilizing a moving average smoothing approach across 25 episodes.

The baseline configuration (No-LLM) maintains relatively stable performance near the 30% mark throughout training, represented by the blue trace. Strategic poisoning under both protected and unprotected conditions demonstrates initial degradation values around 10-15%, which gradually increase through the middle training phase, reaching peak degradation near 35-40% around episode 15 before declining toward episode 25.

The yellow and orange traces, representing different strategic poisoning variants, track closely together with minimal separation, suggesting that causal protection mechanisms provide limited differentiation against this particular attack vector.

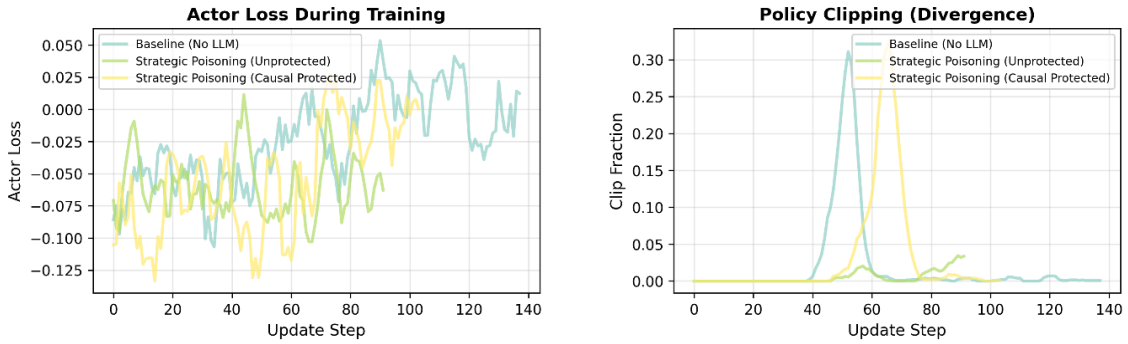

Figure 5.4.2 : Actor Loss and Policy Clipping of the causally safe agent when under attack

The actor loss panel in Figure 5.4.2 (left) tracks policy gradient loss dynamics across 140 update steps for the causally safe agent under multiple experimental conditions. All traces oscillate within a narrow band between approximately -0.025 and 0.050, with baseline (No-LLM) represented in blue, strategic poisoning circumvented shown in green, and both protected and unprotected strategic poisoning variants in yellow and orange respectively.

The high degree of overlap among these traces indicates that adversarial attacks do not substantially alter the optimization landscape as measured by actor loss, suggesting that performance degradation operates through mechanisms not directly captured by this loss metric.

The policy clipping divergence plot in Figure 5.4.2 (right panel) examines the frequency of clipped policy updates, revealing a distinctive distribution pattern.

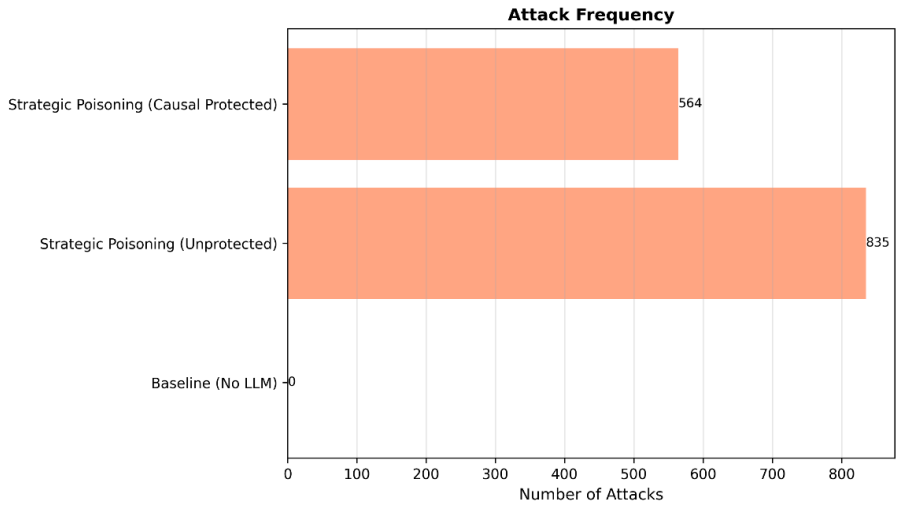

Figure 5.4.3 : Overview of the attack frequency in the Causally Protected system

As per Figure 5.4.3 it can be seen that Strategic poisoning in the unprotected condition receives 835 attacks, represented by the longer bar extending rightward. Strategic poisoning with causal protection experiences 564 attacks, indicated by the shorter bar above it.

The baseline (No-LLM) condition shows zero attacks, as expected for a control configuration. The substantial reduction in attack frequency under causal protection (approximately 32% fewer attacks compared to unprotected) suggests that the causal safety mechanism either prevents certain attack opportunities from arising or filters adversarial interventions before they reach the agent's decision-making process.

This attack frequency differential provides context for interpreting performance comparisons, as the protected agent faces fewer total adversarial interactions despite operating in a comparable threat environment.

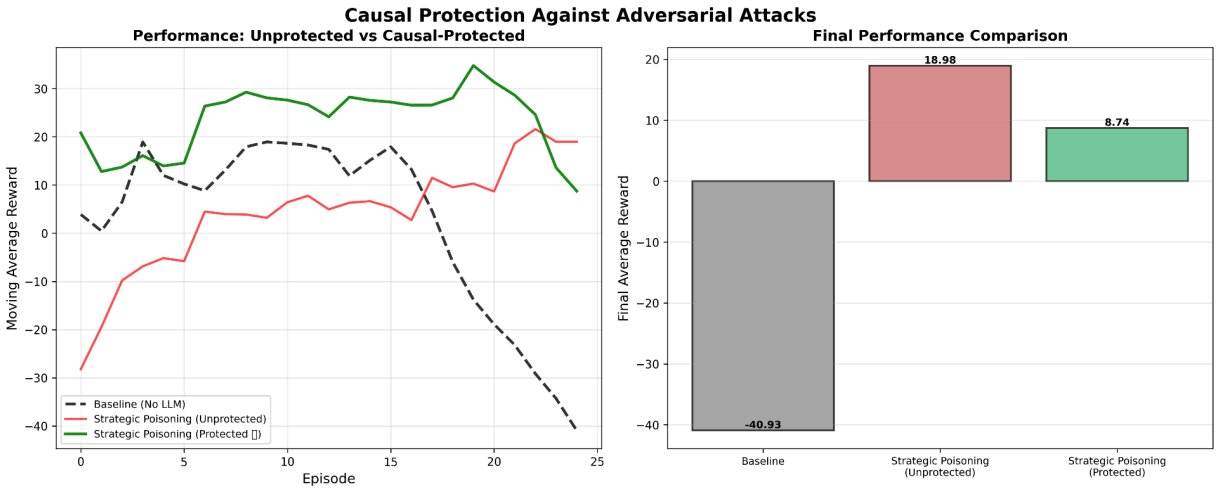

Figure 5.4.4 : Analysis of performance under adversarial attacks in the causally safe agent

In the Figure 5.4.4 the left panel displays raw performance trajectories across 25 episodes for unprotected and causally protected variants of strategic poisoning attacks.

The green plot (circumvented strategic poisoning) begins almost at -5% degradation and steadily increases to nearly 10 percent by episode 10; where it levels off with moderate fluctuations around episode 20 and then drops off drastically; ending at around -45%.

The red trace (protected strategic poisoning, unprotected variant) starts at around -30% and slowly approaches 0% in episode 10; plateauing thereafter.

The black dashed line (protected variant) shows intermediate behavior, which oscillates between -10 and 10 percent by most of the episodes and then dropping by a significant amount in the final episodes.

On the right panel, box plot comparisons of final performance distributions are displayed. The baseline injection condition (gray) indicates a median performance of about -80% with significant negative spread.

Both forms of strategic poisoning (red and green box) show median values in the range of 0-10% with compressed interquartile ranges, demonstrating consistent performance even under adversarial circumstances.

This comparison displays that the mechanisms of causal protection are effective in mitigating the causal failure modes from the unprotected baseline even in cases of the extreme failure mode strategic attacks that compromise the defense layer.

CHAPTER 6
CONCLUSION AND FUTURE WORK

6.1 CONCLUSION

This study examines the application of Large Language Models (LLMs) as strategic consultants for reinforcement learning, and identifying positive and negative performance improvements and additional weaknesses. With NetHack Learning Environment, agents guided by the LLM reached more than 2200 cumulative rewards in 100 episodes- being stable while baseline PPO agents failed to perform consistently after episode 50. Antagonistic testing of this system revealed major flaws: strategic poisoning led to over 300 percent performance decline in late training, and noise and state injection attacks maintained performance declines of over 800 units.

To solve this, a causal protection framework was created, a more attack-resistant model, propensity-weighted filtering and doubly robust estimators were used, decreasing the impact of attacks by approximately 32%; without prior attack information. Although it performed well with most threats, it was poor at identifying adversarial advice that looked like legitimate advice, which is a challenge of identifying semantically coherent yet harmful inputs.

The key contributions of this work are: (1) deciding quantitative baselines of the LLM-guided reinforcement learning in adversarial conditions, (2) outlining a taxonomy of semantic attacks against state descriptions, and (3) substantiating causal inference as a suitable trust calibration mechanism. Generally, the article demonstrates that understanding how and when to doubt an oracle is just as important as using its advice to recover half to two-thirds of performance losses in the face of manipulation and develop stronger, self-assessing AI capabilities.

6.2 Future Work

Future research will concentrate on enhancing the model in two ways. First, it will investigate ensemble approaches for improving LLM application. Second, it will explore adversarial training that incorporates causal feedback to develop a form of policy reserves. Later, the causal framework will be improved through hierarchical modelling; specifically, the framework's current capabilities to assess the level of effect will be improved by taking both high-level strategy and low-level tactic into consideration. Each hierarchy can now be observed as affecting the other instead of each acting independently.

Eventually, research will extend the framework and investigate its varied applications including transfer learning to check if the models apply to new environments, adaption to continuous action spaces (e.g. robotics), and the study of multi-agent foci that involves combining adversarial generative models with honest (causal based or agnostic) model guidance. Ultimately, the framework will be validated in real-world high-stakes situations like automated trading or medical decision support.

BIBLIOGRAPHY

1) Zeng, F., Gan, W., Wang, Y., Liu, N., & Yu, P. S. (2023). Large language models for robotics: A survey. arXiv preprint arXiv:2311.07226. Retrieved from https://arxiv.org/abs/2311.07226
2) Liu, S., et al. (2024). RL-GPT: Integrating reinforcement learning and code-as-policy. arXiv preprint arXiv:2402.19299. Retrieved from https://arxiv.org/abs/2402.19299
3) Carta, T., et al. (2024). Grounding large language models in interactive environments with online reinforcement learning. arXiv preprint arXiv:2302.02662. Retrieved from https://arxiv.org/abs/2302.02662
4) Ahn, M., et al. (2022). Do as I can, not as I say: Grounding language in robotic affordances. arXiv preprint arXiv:2204.01691. Retrieved from https://arxiv.org/abs/2204.01691
5) Liang, J., et al. (2023). Code as policies: Language model programs for embodied control. arXiv preprint arXiv:2209.07753. Retrieved from https://arxiv.org/abs/2209.07753
6) Wang, G., et al. (2023). Voyager: An open-ended embodied agent with large language models. arXiv preprint arXiv:2305.16291. Retrieved from https://arxiv.org/abs/2305.16291
7) Wu, S., et al. (2024). Enhance reasoning for large language models in the game Werewolf. arXiv preprint arXiv:2402.02330. Retrieved from https://arxiv.org/abs/2402.02330
8) Huang, W., et al. (2023). Inner monologue: Embodied reasoning through planning with language models. In Proceedings of the 6th Conference on Robot Learning (PMLR, Vol. 205, pp. 1769–1782). Retrieved from https://proceedings.mlr.press/v205/huang23c.html
9) Sahoo, S. S., et al. (2024). Large language models for biomedicine: Foundations, opportunities, challenges, and best practices. Journal of the American Medical Informatics Association, 31(9), 2114–2124. https://doi.org/10.1093/jamia/ocae074
10) Küttler, H., et al. (2020). The NetHack learning environment. arXiv preprint arXiv:2006.13760. Retrieved from https://arxiv.org/abs/2006.13760
11) Chevalier-Boisvert, M., et al. (2019). BabyAI: A platform to study the sample efficiency of grounded language learning. arXiv preprint arXiv:1810.08272. Retrieved from https://arxiv.org/abs/1810.08272
12) Shridhar, M., et al. (2021). ALFWorld: Aligning text and embodied environments for interactive learning. arXiv preprint arXiv:2010.03768. Retrieved from https://arxiv.org/abs/2010.03768
13) Mees, O., et al. (2022). CALVIN: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. arXiv preprint arXiv:2112.03227. Retrieved from https://arxiv.org/abs/2112.03227
14) Amodei, D., et al. (2016). Concrete problems in AI safety. arXiv preprint arXiv:1606.06565. Retrieved from https://arxiv.org/abs/1606.06565
15) Raji, I. D., & Dobbe, R. (2023). Concrete problems in AI safety, revisited. arXiv preprint arXiv:2401.10899. Retrieved from https://arxiv.org/abs/2401.10899
16) Bhattacharjee, A., et al. (2023). Towards LLM-guided causal explainability for black-box text classifiers. Retrieved from https://api.semanticscholar.org/CorpusID:262459118
17) García, J., & Fernández, F. A. (2015). A comprehensive survey on safe reinforcement learning. Journal of Machine Learning Research, 16(42), 1437–1480. Retrieved from http://jmlr.org/papers/v16/garcia15a.html