

DIPARTIMENTO DI INFORMATICA
UNIVERSITÀ DEGLI STUDI DI SALERNO

Corso di Intelligenza Artificiale

Adaptive Decision Making NPC in Crafter

Architettura HeRoN per Reinforcement Learning

Realizzato da:

DANILO GISOLFI Matricola: 0522502001

VINCENZO MAIELLARO Matricola: 0522502055

Anno Accademico 2025/2026

Abstract

Il progetto HeRoN propone un'architettura innovativa a tre agenti che combina apprendimento per rinforzo e ragionamento basato su Large Language Models (LLM) per controllare in modo intelligente agenti nel gioco Crafter.

La pipeline unisce tre componenti chiave: un DQNAgent basato su Double DQN con replay prioritario per il controllo basato su valore, un Helper che genera sequenze d'azione tramite LLM per la pianificazione a breve termine, e un Reviewer che dà feedback critico e revisioni alle proposte del Helper.

Lo studio copre l'estrazione dello stato attraverso un vettore a 43 dimensioni, la mappatura delle 17 azioni discrete del dominio Crafter, l'implementazione di strategie di checkpointing avanzate e la valutazione comparativa contro baseline DQN tradizionali.

I risultati sperimentali includono metriche di apprendimento dettagliate, analisi approfondite delle policy miste RL+LLM e strumenti avanzati per la visualizzazione e la valutazione degli apprendimenti conseguiti dall'architettura proposta.

Indice

1	Introduzione	6
1.1	Contesto	6
1.2	Motivazione	6
1.3	Obiettivi del Progetto	6
1.3.1	Obiettivi Primari	6
1.3.2	Obiettivi Secondari	7
2	Architettura HeRoN	8
2.1	Panoramica dell'Architettura	8
2.1.1	Diagramma Architettura DQN Baseline	8
2.1.2	Diagramma Architettura HeRoN Completa	8
2.1.3	NPC (Non-Player Character) - Versione Semplificata	9
2.1.4	Helper	10
2.1.5	Reviewer	11
2.2	Gestione del Contesto dell'Helper	11
2.2.1	Struttura della Memoria Conversazionale	11
2.2.2	Gestione Intelligente del Contesto (Token-Aware)	12
2.2.3	Meccanismi di Re-planning e Aggiornamento Contesto	12
2.2.4	Reset Episodio e Pulizia Contesto	13
2.3	Workflow dell'Architettura	13
2.3.1	Fase 1: Decisione di Consultazione	13
2.3.2	Fase 2: Review e Raffinamento (Reviewer)	14
2.3.3	Fase 3: Esecuzione e Re-planning	14
2.4	Vantaggi dell'Architettura	14
2.5	Sfide dell'Integrazione	15
3	Environment Crafter	16
3.1	Introduzione a Crafter	16
3.1.1	Caratteristiche Principali	16
3.2	Meccaniche di Gioco	16
3.2.1	Obiettivi di Sopravvivenza	16
3.2.2	Sistema di Progressione	17
3.3	Spazio di Stati	17
3.3.1	Spazio delle Azioni	18
3.3.2	Sistema di Reward	18

4 Metodologia di Implementazione	20
4.1 Introduzione	20
4.2 Panoramica del Processo	20
4.3 NPC (DQN Agent)	20
4.3.1 Architettura della Rete Neurale	20
4.4 Helper (LLM) e Prompt Design	23
4.5 Generazione del Dataset per il Reviewer	25
4.5.1 Fase 5: Fine-tuning del Reviewer	26
4.6 Training Integrato HeRoN	27
4.7 Valutazione delle Prestazioni	29
4.7.1 Fase 8: Analisi e Ottimizzazione	29
4.7.2 Tuning degli Iperparametri	29
4.7.3 Tuning e Configurazioni del Reviewer (T5)	30
4.8 Estensione: Fine-tuning del Reviewer tramite RL	30
5 Risultati Sperimentali	32
5.1 Introduzione	32
5.2 Setup Sperimentale	32
5.3 Configurazioni Testate	32
5.4 Confronto tra Configurazioni	33
5.4.1 DQN Baseline	33
5.4.2 DQN+Helper	34
5.4.3 HeRoN	36
5.4.4 Confronti Diretti	38
5.4.5 Coverage Achievement	38
5.4.6 Reward Cumulativo	39
5.4.7 Analisi Qualitativa	39
5.5 Conclusioni	39
5.5.1 Success Rate per Achievement	47
5.5.2 Reward Cumulativo - Dettaglio	47
5.5.3 Analisi della Convergenza	47
5.5.4 Analisi del Numero di Azioni per Sequenza	49
5.5.5 Sessioni di Addestramento del NPC	49
5.5.6 Dimostrazione dell'Abilità del NPC nello Svolgere i Task	50
5.6 Analisi Comparativa Finale	51
5.6.1 Riepilogo Metriche Chiave	51
5.6.2 Conclusioni Finali	51
6 Conclusioni	52
6.1 Sintesi del Lavoro Svolto	52
6.2 Risultati Principali	52
6.2.1 Performance Quantitative	52
6.3 Efficacia dei Componenti e Sfide Affrontate	52
6.3.1 Challenge 1: Sparsità del Reward	53
6.3.2 Challenge 2: Gestione Situazioni Critiche	53
6.3.3 Challenge 3: LLM Hallucinations e Action Typos	53
6.3.4 Limitazioni	54
6.3.5 Lavori Futuri	54
6.3.6 Considerazioni Finali	55

Elenco delle figure

5.1	Dashboard multi-metrica DQN Baseline.	33
5.2	Heatmap achievement DQN Baseline.	34
5.3	Curve di apprendimento collect_wood - DQN Baseline.	34
5.4	Dashboard multi-metrica DQN+Helper.	35
5.5	Heatmap achievement DQN+Helper.	35
5.6	Curve di apprendimento collect_wood - DQN+Helper.	36
5.7	Dashboard multi-metrica HeRoN.	37
5.8	Heatmap achievement HeRoN.	37
5.9	Curve di apprendimento collect_wood - HeRoN.	38
5.10	Confronto curve di apprendimento per collect_wood.	38
5.11	Dashboard multi-metrica DQN+Helper. Il confronto mostra miglioramento su tutte le metriche rispetto al DQN Baseline: reward più concentrato, coverage degli achievement più ampio, efficienza temporale superiore.	40
5.12	Heatmap della distribuzione degli achievement per DQN+Helper. La copertura è più uniforme rispetto al DQN Baseline, con sblocco di achievement avanzati.	40
5.13	Scatter plot dell'efficienza temporale - DQN+Helper: reward per step vs episodio. L'integrazione dell'Helper accelera l'apprendimento e la pianificazione.	41
5.14	Curve di apprendimento per collect_wood - DQN+Helper. Plateau raggiunto più rapidamente rispetto al DQN Baseline.	41
5.15	Curve di apprendimento per collect_sapling - DQN+Helper. Success rate superiore e maggiore stabilità.	42
5.16	Curve di apprendimento per place_table - DQN+Helper. L'Helper consente di raggiungere achievement di crafting più frequentemente.	42
5.17	Curve di apprendimento per place_plant - DQN+Helper. Strategie di farming più efficaci rispetto al baseline.	43
5.18	Curve di apprendimento per defeat_zombie - DQN+Helper. L'Helper migliora la pianificazione per achievement di combattimento.	43
5.19	Curve di apprendimento per collect_drink - DQN+Helper. Gestione risorse vitali appresa più rapidamente.	44
5.20	Curve di apprendimento per wake_up - DQN+Helper. Gestione ciclo giorno/notte più efficace.	44
5.21	Dashboard multi-metrica HeRoN. Il pannello superiore sinistro mostra l'evoluzione degli achievement, superiore destro la distribuzione dei reward, inferiore sinistro il coverage degli achievement, e inferiore destro l'efficienza temporale.	45

5.22 Dashboard multi-metrica DQN Baseline per confronto. Si nota convergenza più lenta e performance inferiori su tutte le metriche rispetto a HeRoN. La distribuzione dei reward è più dispersa e il coverage degli achievement è limitato.	46
5.23 Progressione achievement durante training con medie mobili - HeRoN. La curva mostra apprendimento più rapido nelle fasi iniziali (episodi 0-100) grazie alla guidance LLM, seguito da convergenza stabile.	48
5.24 Progressione achievement durante training con medie mobili - DQN Baseline. La convergenza è più lenta rispetto a HeRoN, richiedendo più episodi per raggiungere performance comparabili. La curva mostra maggiore varianza iniziale.	48

Elenco delle tabelle

4.1	Confronto tra modelli LLM testati per l'Helper. Qwen3-4B-2507 è il migliore per conformità e coerenza	24
4.2	Versioni dei prompt Helper LLM	24
4.3	Esempi esplicativi dei prompt Helper LLM utilizzati	25
4.4	Impatto del numero di azioni per sequenza	27
4.5	Configurazioni testate per il tuning del Reviewer (T5)	30
4.6	Workflow Fine-Tuning Reviewer RL	31
5.1	Parametri di training	32
5.2	Metriche principali DQN Baseline	33
5.3	Metriche principali DQN+Helper	34
5.4	Metriche principali HeRoN	36
5.5	Coverage degli achievement	38
5.6	Reward cumulativo per episodio (ultimi 100 episodi)	39
5.7	Success rate per tutti gli achievement (DQN: 300 episodi, DQN+Helper: 300 episodi, HeRoN: 301 episodi)	47
5.8	Reward cumulativo per episodio (ultimi 100 episodi)	47
5.9	Velocità di convergenza	49
5.10	Performance per configurazione di training (ultimi 100 episodi)	49
5.11	Success rate per categoria di task	50
5.12	Riepilogo delle metriche chiave per configurazione	51
6.1	Impatto delle soluzioni implementate	54

Capitolo 1

Introduzione

1.1 Contesto

Questo progetto fa parte del campo del Reinforcement Learning applicato ai videogiochi, un'area di ricerca in rapida crescita che mira a creare agenti intelligenti capaci di imparare strategie ottimali interagendo con ambienti di gioco.

I videogiochi moderni, soprattutto quelli open-world e di sopravvivenza, presentano sfide complesse che richiedono agli agenti di prendere decisioni strategiche a lungo termine, gestire risorse limitate e adattarsi a situazioni dinamiche. Questi ambienti sono perfetti per testare e validare nuove idee di intelligenza artificiale.

1.2 Motivazione

L'architettura HeRoN (Helper-Reviewer-NPC) è un approccio innovativo che combina il Reinforcement Learning tradizionale con il ragionamento dei Large Language Model (LLM). Questa architettura è stata inizialmente validata in environment di tipo JRPG (Japanese Role-Playing Game) a turni, dimostrando la sua efficacia nel migliorare le prestazioni degli agenti RL attraverso suggerimenti strategici forniti da modelli linguistici.

La sfida principale di questo progetto è stata estendere e testare HeRoN in un contesto molto diverso: il gioco Crafter, un open-world di sopravvivenza che richiede pianificazione a lungo termine, gestione delle risorse e adattamento dinamico.

1.3 Obiettivi del Progetto

Gli obiettivi principali del lavoro sono i seguenti:

1.3.1 Obiettivi Primari

- **Adattamento dell'architettura HeRoN:** L'architettura HeRoN viene estesa dall'environment JRPG a turni all'environment Crafter, un survival game open-world in tempo continuo.
- **Fine-tuning del Reviewer:** Il componente Reviewer viene adattato ai nuovi task specifici di Crafter, con generazione di un dataset appropriato e addestramento del modello per fornire feedback efficaci nel contesto del survival game.

- **Modifica del Helper:** Il comportamento del componente Helper viene modificato affinché vengano generate sequenze di azioni coerenti (3-5 azioni) anziché singole decisioni, permettendo una pianificazione più strategica.
- **Implementazione dell'NPC:** Viene sviluppato un agente di Reinforcement Learning basato sull'algoritmo Deep Q-Network (DQN) ottimizzato per le 17 azioni disponibili in Crafter e il suo spazio di stati a 43 dimensioni.
- **Valutazione delle prestazioni:** Le prestazioni dell'architettura HeRoN completa vengono valutate quantitativamente rispetto a baseline tradizionali, misurando il numero di achievement sbloccati nei 22 obiettivi disponibili in Crafter.

1.3.2 Obiettivi Secondari

- Analisi del numero ottimale di azioni da suggerire per ciascuna chiamata del Helper.
- Studio dell'impatto del reward shaping sulle prestazioni dell'agente.
- Implementazione di meccanismi di re-planning che interrompono le sequenze di azioni in situazioni critiche (salute bassa, achievement sbloccati).

Capitolo 2

Architettura HeRoN

2.1 Panoramica dell'Architettura

HeRoN (Helper-Reviewer-NPC) è un'architettura multi-agente che combina Reinforcement Learning e Large Language Model per migliorare il processo decisionale di agenti intelligenti in ambienti interattivi. L'idea di base è unire la capacità del Reinforcement Learning di ottimizzare strategie attraverso prove ed errori, il ragionamento semantico e la conoscenza generale dei Large Language Model, e un meccanismo di feedback iterativo per migliorare i suggerimenti.

2.1.1 Diagramma Architettura DQN Baseline

Prima di descrivere l'architettura completa HeRoN, viene presentata l'architettura baseline DQN utilizzata come riferimento per il confronto con l'integrazione LLM.

Flusso Operativo DQN:

1. **Percezione:** Environment → State Extraction (43-dim vector)
2. **Decisione:** DQN Network → Q-values → ϵ -greedy selection
3. **Azione:** Execute action a_t , observe r_t, s_{t+1}
4. **Memorizzazione:** Salva $(s_t, a_t, r_{shaped}, s_{t+1}, done)$ in Prioritized Replay
5. **Apprendimento:** Sample batch → compute TD-loss → update DQN weights
6. **Stabilizzazione:** Ogni 100 steps, copia DQN weights → Target Network

L'architettura DQN baseline apprende esclusivamente tramite interazione diretta con l'ambiente, senza supporto esterno.

2.1.2 Diagramma Architettura HeRoN Completa

L'architettura HeRoN rappresenta un'estensione del DQN baseline, con l'aggiunta di due componenti LLM per la guida strategica, e l'utilizzo di un meccanismo di threshold decay che bilancia l'intervento tra LLM e RL.

Meccanismo Chiave - Threshold Decay:

Il threshold θ controlla la probabilità di consultare LLM vs. usare DQN autonomo:

$$\theta(e) = \max(0, 1.0 - 0.01 \times e) \quad (2.1)$$

dove e è il numero dell'episodio corrente. Questo garantisce:

- **Episodi 0-100:** $\theta: 1.0 \rightarrow 0.0$, transizione graduale da 100% LLM a 0% LLM
- **Episodi 100+:** $\theta = 0$, DQN completamente autonomo
- **Vantaggio:** Nelle fasi iniziali, l'intervento LLM supporta il processo decisionale, mentre nelle fasi successive il DQN opera in modo autonomo

Flusso Completo (Esempio con LLM Path):

1. Environment genera stato \rightarrow State Extraction (43-dim)
2. Threshold check: $\text{random}(0.73) > \text{threshold}(0.65) \rightarrow \text{LLM Path}$
3. Helper riceve state description \rightarrow genera `[move_right]`, `[do]`, `[move_left]`, `[do]`, `[noop]`
4. Reviewer analizza \rightarrow feedback: *"Health low, prioritize eating"*
5. Helper re-query con feedback \rightarrow sequenza raffinata: `[eat_plant]`, `[sleep]`, `[move_right]`, `[do]`, `[noop]`
6. Action Executor esegue `[eat_plant]` $\rightarrow (s_1, r_1, info_1)$
7. Salva $(s_0, eat_plant, r_1, s_1, done)$ in Prioritized Replay
8. Monitor: achievement unlocked? $\rightarrow \text{SÌ} \rightarrow$ interrompi sequenza, nuova query Helper
9. DQN training: sample batch \rightarrow compute loss \rightarrow update weights

L'architettura HeRoN integra i vantaggi del Reinforcement Learning (apprendimento da esperienza) e dei Large Language Model (conoscenza a priori e ragionamento strategico), con una transizione graduale verso l'autonomia dell'agente.

L'architettura HeRoN è composta da tre componenti principali che interagiscono in modo coordinato:

2.1.3 NPC (Non-Player Character) - Versione Semplificata

L'NPC è l'agente che gioca in Crafter usando il Reinforcement Learning. In questo progetto, l'NPC usa l'algoritmo Deep Q-Network (DQN), che funziona così:

- **Architettura:** DQN usa una seconda rete (target network) per rendere l'apprendimento più stabile.
- **Replay Buffer:** Salva le esperienze passate e dà più importanza a quelle utili, con spazio per 10.000 eventi.
- **Parametri principali:**
 - $\alpha = 0.6$: priorità alle esperienze importanti
 - $\beta = 0.4$ che cresce fino a 1.0: corregge il campionamento
 - $\gamma = 0.99$: importanza delle ricompense future

– $\epsilon = 1.0$ che scende fino a 0.05: probabilità di provare azioni nuove

- **Input:** Stato dell'ambiente (43 numeri)
- **Output:** Valori Q per 17 azioni possibili

In pratica, l'NPC osserva lo stato, sceglie un'azione, riceve una ricompensa e aggiorna la sua memoria per imparare a giocare meglio nel tempo.

2.1.4 Helper

Il componente Helper è un Large Language Model utilizzato in modalità zero-shot che fornisce suggerimenti strategici all'NPC. Nel progetto HeRoN per Crafter, l'Helper è implementato utilizzando un LLM locale (Qwen3-4B-2507) attraverso LM Studio con le seguenti caratteristiche:

- **Generazione di sequenze di azioni:** Diversamente dall'implementazione originale che suggeriva singole azioni, l'Helper in questo progetto genera sequenze di 3-5 azioni coerenti da eseguire una dopo l'altra.
- **Contestualizzazione:** L'Helper riceve informazioni dettagliate sullo stato corrente del gioco:
 - Inventario del giocatore (16 item)
 - Posizione corrente
 - Statistiche vitali (salute, cibo, acqua)
 - Achievement sbloccati (22 possibili)
- **Prompt Engineering:** Il prompt è stato specificamente progettato per Crafter e include:
 - Descrizione del contesto di gioco
 - Stato corrente dell'agente
 - Lista delle azioni disponibili
 - Richiesta di generare una sequenza strategica

L'Helper risponde con una sequenza di azioni nel formato:

```
[azione_1], [azione_2], [azione_3], [azione_4], [azione_5]
```

Ad esempio:

```
[move_right], [do], [move_left], [do], [noop]
```

2.1.5 Reviewer

Il componente Reviewer è un LLM fine-tuned (basato su T5) che valuta i suggerimenti forniti dall'Helper e genera feedback per migliorarli. Il Reviewer è stato addestrato specificamente per il contesto di Crafter utilizzando un dataset generato attraverso:

1. Raccolta di stati dell'environment durante sessioni di gioco
2. Generazione di suggerimenti dall'Helper per ciascuno stato
3. Annotazione manuale o semi-automatica di feedback correttivi
4. Fine-tuning del modello T5 su coppie (suggerimento, feedback)

Il dataset contiene circa 2,500 esempi raccolti da 50 episodi di gioco.

Il Reviewer analizza:

- Coerenza della sequenza di azioni suggerite
- Appropriatezza rispetto allo stato corrente
- Potenziali rischi o inefficienze
- Priorità strategiche (es. sopravvivenza vs. progressione)
- Fornisce feedback strutturato che usiamo per ri-interrogare l'Helper con più informazioni.

2.2 Gestione del Contesto dell'Helper

Il componente Helper mantiene uno stato conversazionale persistente durante ogni episodio per costruire un contesto ricco e coerente. La gestione del contesto è un aspetto critico dell'architettura, in quanto influenza direttamente la qualità e la coerenza dei suggerimenti generati.

2.2.1 Struttura della Memoria Conversazionale

L'Helper mantiene una cronologia di messaggi (`message_history`) che accumula:

- **Prompt iniziali:** Descrizioni dello stato di gioco e richieste di azioni
- **Risposte LLM:** Sequenze di azioni generate precedentemente
- **Feedback del Reviewer:** Suggerimenti strategici per migliorare i piani
- **Contesto di gioco:** Posizione, inventario, achievement sbloccati

La cronologia consente al LLM di mantenere coerenza logica tra azioni successive e di comprendere l'evoluzione dello stato di gioco all'interno dell'episodio.

2.2.2 Gestione Intelligente del Contesto (Token-Aware)

L'Helper implementa un sistema di gestione intelligente del contesto per prevenire l'overflow della finestra di contesto del modello LLM (Qwen3-4B-2507 ha 8192 token di limite):

1. **Monitoraggio token:** Il Helper conta il numero di token nella cronologia usando il tokenizer Qwen2.5 (compatibile con Qwen3)
2. **Soglia di sicurezza:** Quando il contesto raggiunge 6500 token (80% del limite), viene attivato un reset intelligente per evitare crash e risposte vuote
3. **Reset con riassunto episodio:** Invece di scartare tutto il contesto, il Helper genera un **riassunto condensato** che include:
 - Numero di step eseguiti nell'episodio
 - Reward totale accumulato
 - Lista degli achievement sbloccati
 - Feedback recente del Reviewer (se disponibile)
 - Descrizione dello stato di gioco corrente

Questo riassunto diventa il nuovo inizio della cronologia, preservando informazioni strategiche critiche.

Vantaggi del reset intelligente:

- **Continuità strategica:** Il modello comprende il progresso episodico complessivo
- **Efficienza token:** Riduce i token inutili mantenendo informazioni essenziali
- **Riduzione allucinazioni:** Contesto pulito riduce risposte errate o non coerenti
- **Maggiore lunghezza episodio:** Consente episodi più lunghi senza crash

2.2.3 Meccanismi di Re-planning e Aggiornamento Contesto

Durante l'esecuzione di una sequenza di azioni, il Helper monitora determinati eventi per aggiornare intelligentemente il contesto:

Trigger di Re-query (Interruzione Sequenza):

- **Achievement sbloccato:** Quando il giocatore sblocca un nuovo achievement, il Helper riceve una nuova query con il contesto aggiornato che include il nuovo achievement nel set di quelli sbloccati
- **Salute critica ($health \leq 5$):** Se la salute scende sotto soglia critica, la sequenza viene interrotta e il Helper viene re-interrogato per suggerire azioni di emergenza (mangiare, bere, dormire)
- **Salute bassa ($health < 30\%$):** Se la salute è bassa ma non critica, il Helper viene ri-consultato per bilanciare l'esplorazione con la gestione della sopravvivenza

- **Risorsa completamente consumata:** Se una risorsa chiave (legno, pietra, carbone) raggiunge 0, viene attivata una nuova query per raccoglierla prioritariamente

Aggiornamento dello Stato Episodio:

Ad ogni passo, il Helper aggiorna il suo tracciamento interno:

```
helper.update_episode_progress(
    achievements=nuovi_achievement_sbloccati,
    step_count=numero_step_corrente,
    reward=reward_episodio_accumulato,
    reviewer_feedback=ultimo_feedback_reviewer
)
```

Questi dati vengono usati durante il reset del contesto per generare il riassunto episodico. Questo assicura che il LLM comprenda il progresso complessivo anche dopo un reset di contesto.

2.2.4 Reset Episodio e Pulizia Contesto

All'inizio di ogni nuovo episodio, il Helper esegue una pulizia completa:

- Cancellazione della cronologia messaggi (`message_history = []`)
- Reset del tracciamento achievement episodio
- Azzeramento del feedback Reviewer recente
- Pulizia della cronologia delle sequenze (usata per rilevare loop)

Questo previene l'accumulo di contesto da episodi precedenti e garantisce che il LLM inizi ogni episodio con uno stato cognitivo "fresco".

Esempio di reset a episodio 5:

```
[Episode 5] Conversation history cleared for new episode
[Helper] Token overflow detected: 7200/8192 tokens
[Helper] Current context: 6500, New prompt: 700
[Helper] Performing SMART RESET with episode summary...
[Helper] Context reset complete: 1200 tokens (saved 5300 tokens)
[Helper] Summary includes: 5 achievements, step 342
```

2.3 Workflow dell'Architettura

Il workflow di HeRoN durante il training si articola in queste fasi:

2.3.1 Fase 1: Decisione di Consultazione

Ad ogni step dell'episodio, l'architettura decide se consultare i componenti LLM basandosi su una soglia dinamica θ .

Algorithm 1 Decisione di consultazione LLM

```

if random() >  $\theta$  AND episode < 600 then
    Procedi con workflow LLM
else
    Usa solo DQN:  $a = \arg \max_a Q(s, a)$ 
end if

```

Quando decidiamo di consultare l'LLM, l'Helper genera una sequenza di azioni basandosi sullo stato corrente.

La soglia θ decresce linearmente da 1.0 a 0.0 nel corso di 100 episodi:

$$\theta_t = \max(0, \theta_0 - 0.01 \cdot \text{episode})$$

In questo modo, l'NPC si affida di più ai suggerimenti LLM all'inizio dell'allenamento, per poi ridurre questa dipendenza man mano che l'agente RL migliora.

2.3.2 Fase 2: Review e Raffinamento (Reviewer)

Se il Reviewer è disponibile, valuta la sequenza proposta e fornisce un feedback. L'Helper usa il feedback per migliorare la sequenza e ne crea una seconda versione. Infine, usiamo la sequenza migliorata se il Reviewer ha dato feedback, altrimenti la prima.

2.3.3 Fase 3: Esecuzione e Re-planning

La sequenza di azioni viene eseguita una dopo l'altra, ma può essere interrotta in caso di eventi importanti:

- **Achievement sbloccato:** Nuova consultazione LLM con contesto aggiornato
- **Salute critica** ($health \leq 5$): Fallback immediato a DQN per azioni di sopravvivenza
- **Salute bassa** ($health < 30\%$): Ri-query del sistema per gestione prioritaria della salute

2.4 Vantaggi dell'Architettura

L'architettura HeRoN combina RL e LLM offrendo:

- **Conoscenza a priori:** LLM accelera l'apprendimento con conoscenze generali
- **Ragionamento strategico:** Pianificazione di azioni coerenti a lungo termine
- **Adattabilità:** Unisce esplorazione RL e suggerimenti LLM per nuove situazioni
- **Interpretabilità:** Sequenze di azioni analizzabili per capire la strategia
- **Raffinamento iterativo:** Helper e Reviewer migliorano la qualità dei suggerimenti

2.5 Sfide dell'Integrazione

Le principali difficoltà nell'integrazione RL-LLM sono:

- **Overhead computazionale:** LLM più costosi rispetto al DQN
- **Parsing delle risposte:** Gestione di risposte errate o non valide
- **Bilanciamento:** Equilibrio tra dipendenza da LLM e autonomia RL
- **Consistenza:** Garantire sequenze eseguibili e coerenti

Capitolo 3

Environment Crafter

3.1 Introduzione a Crafter

Crafter è un environment di ricerca per Reinforcement Learning, ispirato a Minecraft ma più semplice e controllato. Serve a valutare le capacità degli agenti RL, dalla sopravvivenza base alla progressione tecnologica.

3.1.1 Caratteristiche Principali

- **Open-world 2D:** Mondo generato proceduralmente con terreni vari
- **Survival game:** Raccolta risorse, crafting e sopravvivenza
- **Osservazioni visive:** Frame RGB $64 \times 64 \times 3$
- **22 Achievement:** Obiettivi progressivi che testano varie abilità
- **Episodi limitati:** Durata massima di 10,000 step per episodio

3.2 Meccaniche di Gioco

3.2.1 Obiettivi di Sopravvivenza

Il giocatore deve gestire tre statistiche vitali:

- **Salute (Health):** Diminuisce se attaccato dai mostri, a zero termina l'episodio
- **Cibo (Food):** Diminuisce col tempo; se a zero, la salute cala
- **Acqua (Water):** Diminuisce col tempo; se a zero, la salute cala

Per sopravvivere, il giocatore deve:

1. Raccogliere cibo (piante, animali)
2. Bere acqua esplorando il mondo
3. Dormire per rigenerare salute
4. Evitare o combattere i mostri

3.2.2 Sistema di Progressione

Il sistema di progressione tecnologica include:

1. **Raccolta base:** Legno, pietra
2. **Costruzione strumenti:** Tavolo di lavoro, fornace
3. **Strumenti di pietra:** Piccone, spada
4. **Strumenti di ferro:** Estrazione ferro e crafting avanzato

Ogni livello sblocca nuove azioni e obiettivi.

3.3 Spazio di Stati

Nel progetto usiamo una rappresentazione strutturata a 43 dimensioni per migliorare efficienza, interpretabilità, apprendimento e compatibilità con LLM:

Inventario (16 dimensioni) Conteggio degli oggetti:

```
[wood, stone, coal, iron, diamond, sapling,
wood_pickaxe, stone_pickaxe, iron_pickaxe,
wood_sword, stone_sword, iron_sword,
drink, food, health_potion, arrow]
```

Posizione e Orientamento (2 dimensioni)

- Coordinata X (normalizzata)
- Coordinata Y (normalizzata)

Statistiche Vitali (3 dimensioni)

- Salute (0-9)
- Cibo (0-9)
- Acqua (0-9)

Achievement (22 dimensioni) Vettore binario degli achievement sbloccati:

```
[collect_wood, collect_stone, collect_coal,
collect_iron, collect_diamond, place_table,
place_furnace, place_plant, place_stone,
defeat_zombie, defeat_skeleton, eat_cow,
eat_plant, drink_water, make_wood_pickaxe,
make_stone_pickaxe, make_iron_pickaxe,
make_wood_sword, make_stone_sword,
make_iron_sword, sleep, wake_up]
```

3.3.1 Spazio delle Azioni

Crafter prevede 17 azioni discrete:

Movimento (4 azioni)

- move_left, move_right, move_up, move_down

Interazione (2 azioni)

- do (azione contestuale), sleep (rigenera salute, se su erba di notte)

Posizionamento (4 azioni)

- place_stone, place_table, place_furnace, place_plant

Crafting (6 azioni)

- make_wood_pickaxe, make_stone_pickaxe, make_iron_pickaxe,
- make_wood_sword, make_stone_sword, make_iron_sword

Nessuna Azione (1 azione)

- noop (nessuna azione)

3.3.2 Sistema di Reward

Reward Nativo (Sparse)

Crafter fornisce un reward sparso basato sugli achievement:

$$r_{\text{native}} = \begin{cases} +1 & \text{se achievement sbloccato} \\ 0 & \text{altrimenti} \end{cases}$$

Questo reward è estremamente sparso: in un episodio tipico, il giocatore può sbloccare 0-5 achievement su 22 possibili.

Reward Shaping (Dense)

Per facilitare l'apprendimento, viene implementato un sistema di reward shaping che fornisce segnali più frequenti:

$$r_{\text{shaped}} = r_{\text{native}} + r_{\text{resources}} + r_{\text{health}} + r_{\text{tier}} + r_{\text{tools}} \quad (3.1)$$

$$r_{\text{resources}} = 0.1 \times \# \text{ nuove risorse raccolte} \quad (3.2)$$

$$r_{\text{health}} = 0.05 \times \Delta_{\text{health}} \quad (\text{se positivo}) \quad (3.3)$$

$$r_{\text{tier}} = 0.05 \times \Delta_{\text{tier}} \quad (\text{progressione tecnologica}) \quad (3.4)$$

$$r_{\text{tools}} = 0.02 \times \# \text{ nuovi strumenti crafted} \quad (3.5)$$

Il reward shaping mantiene i seguenti principi:

- Non altera gli ottimi della policy (bonus solo per progressi effettivi)
- Mantiene lo stesso ordine di grandezza del reward nativo
- Fornisce feedback più denso durante l'esplorazione iniziale

Dipendenze tra Achievement

Molti achievement hanno dipendenze implicite:

```
collect_wood -> make_wood_pickaxe ->
collect_stone -> make_stone_pickaxe ->
collect_iron -> place_furnace ->
make_iron_pickaxe -> collect_diamond
```

Questa struttura gerarchica richiede all'agente di apprendere sequenze di azioni complesse e pianificazione a lungo termine.

Capitolo 4

Metodologia di Implementazione

4.1 Introduzione

Questo capitolo descrive la metodologia utilizzata per sviluppare e valutare l'architettura HeRoN nel dominio Crafter. Viene fornita una panoramica delle fasi principali (implementazione dell'NPC, integrazione LLM, generazione del dataset per il Reviewer, fine-tuning e training integrato), seguita da dettagli tecnici e protocolli sperimentali.

Le sezioni sono organizzate per guidare il lettore dalla teoria all'implementazione pratica, passando per la progettazione dei moduli, la raccolta dati e la valutazione sperimentale. Ogni tabella e algoritmo è presentato subito dopo la relativa spiegazione, per garantire coerenza e leggibilità.

4.2 Panoramica del Processo

Di seguito vengono sintetizzate le fasi principali del flusso di lavoro. Le sezioni successive sviluppano ciascun punto in dettaglio.

1. Implementazione e addestramento del NPC (DQN)
2. Progettazione e integrazione dell'Helper (LLM)
3. Generazione del dataset per il Reviewer
4. Fine-tuning supervised del Reviewer (T5)
5. Estensione: fine-tuning tramite Reinforcement Learning (PPO)
6. Training integrato dell'architettura HeRoN e valutazione

4.3 NPC (DQN Agent)

4.3.1 Architettura della Rete Neurale

L'agente DQN è stato progettato con una rete neurale feedforward composta da:

- **Input Layer:** 43 neuroni (corrispondenti alla dimensione dello stato)
- **Hidden Layer 1:** 256 neuroni, attivazione ReLU, Dropout(0.1)

- **Hidden Layer 2:** 256 neuroni, attivazione ReLU, Dropout(0.1)
- **Hidden Layer 3:** 128 neuroni, attivazione ReLU
- **Output Layer:** 17 neuroni (Q-values, uno per ciascuna azione)

Implementazione Double DQN

Per evitare la sovrastima dei Q-values, sono state implementate due reti distinte:

- **Policy Network:** aggiornata ad ogni step, utilizzata per selezionare le azioni
- **Target Network:** aggiornata periodicamente, utilizzata per calcolare i target Q-values

Questa distinzione è essenziale per la stabilità dell'apprendimento, come approfondito nella sezione algoritmi RL.

Algoritmi RL: Double DQN, PER, TD-error, Backpropagation

L'addestramento dell'agente DQN si basa su alcuni concetti fondamentali dell'apprendimento automatico, che spieghiamo qui in modo semplice:

- **Funzione di perdita Q (Errore di previsione):**

La funzione di perdita misura la differenza tra la previsione del valore di un'azione in uno stato e il valore ideale atteso. La formula è:

$$L = \mathbb{E}[(Q(s, a) - y)^2]$$

Dove:

- $Q(s, a)$ rappresenta la stima del valore dell'azione a nello stato s .
- y è il valore ideale, calcolato come:

$$y = r + \gamma \max_{a'} Q(s', a')$$

- r è il premio (reward) ottenuto dopo l'esecuzione dell'azione.
- γ è un numero tra 0 e 1 che indica il peso attribuito ai premi futuri (più è vicino a 1, maggiore è l'importanza del lungo termine).
- s' è lo stato successivo dopo l'azione.
- $\max_{a'} Q(s', a')$ indica la migliore stima tra tutte le azioni possibili nel nuovo stato.

- **Double DQN:**

Per evitare che l'agente sia troppo ottimista nelle sue previsioni, si usano due reti neurali diverse:

- La "policy network" sceglie le azioni.
- La "target network" serve solo per calcolare il valore ideale y .

La formula diventa:

$$y = r + \gamma Q'(s', \arg \max_{a'} Q(s', a'))$$

Dove Q' è la rete target e $\arg \max$ indica l'azione migliore secondo la policy network.

- **Prioritized Experience Replay (PER):**

L'agente impara rivedendo le sue esperienze passate. Non tutte le esperienze sono uguali: quelle dove ha sbagliato di più sono più utili per imparare. Si assegna una "priorità" a ogni esperienza usando questa formula:

$$p_i = |\delta_i| + \epsilon$$

Dove:

- δ_i è l'errore di previsione (TD-error):

$$\delta_i = r + \gamma \max_{a'} Q_{target}(s', a') - Q(s, a)$$

Più è grande, più l'agente ha sbagliato e più deve imparare da quell'esperienza.

- ϵ è un piccolo numero che serve solo a evitare problemi matematici.

- **Backpropagation (Aggiornamento dei pesi):**

L'agente aggiorna i "pesi" della rete neurale (cioè i suoi parametri interni) per ridurre l'errore. La regola di aggiornamento è:

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} L$$

Dove:

- θ sono i pesi della rete.
- α è la velocità di apprendimento (learning rate).
- $\nabla_{\theta} L$ è il gradiente, cioè la direzione in cui bisogna cambiare i pesi per ridurre l'errore.

In sintesi: l'agente prova azioni, riceve premi, aggiorna le sue previsioni e impara soprattutto dagli errori più grandi, migliorando così la sua capacità di prendere decisioni nel tempo.

L'aggiornamento della target network avviene ogni $C = 1000$ step tramite soft update:

$$\theta_{target} \leftarrow \tau \theta_{policy} + (1 - \tau) \theta_{target}$$

con $\tau = 0.001$.

Prioritized Experience Replay

Il replay buffer utilizza un campionamento prioritario per migliorare l'efficienza dell'apprendimento:

1. **Calcolo priorità:** Per ogni transizione, la priorità si basa sul TD-error:

$$p_i = |\delta_i|^{\alpha} + \epsilon$$

dove $\delta_i = r + \gamma \max_{a'} Q_{target}(s', a') - Q(s, a)$

2. **Sampling:** La probabilità di selezionare la transizione i è:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

3. **Importance Sampling:** Per correggere il bias introdotto dal campionamento, i pesi sono:

$$w_i = \left(\frac{1}{N \cdot P(i)} \right)^\beta$$

Training Baseline DQN

Prima dell'integrazione dei moduli LLM, è stato addestrato un agente DQN baseline con i seguenti parametri:

- **Episodi:** 1000
- **Max steps per episodio:** 1000
- **Epsilon decay:** $\epsilon = 1.0 \rightarrow 0.05$ in 800 episodi
- **Learning rate:** $\alpha = 0.0001$
- **Batch size:** 64

Questo baseline costituisce il riferimento per valutare l'efficacia dell'integrazione con i moduli LLM.

4.4 Helper (LLM) e Prompt Design

La progettazione del modulo Helper e dei prompt è stata fondamentale per ottenere sequenze di azioni coerenti e strategiche. Le tabelle seguenti mostrano la selezione dei modelli e l'evoluzione dei prompt.

Setup LM Studio

LM Studio è stato configurato per servire il modello Qwen3-4B-2507:

- Server locale su `http://127.0.0.1:1234`
- API compatibile con OpenAI
- Temperatura: 0.7 (bilanciamento tra creatività e coerenza)
- Max tokens: 150 (sufficiente per sequenze di 3-5 azioni)
- Context window: 8192 tokens (gestione conversazioni lunghe)
- Tokenizer: Qwen2.5-7B-Instruct (per conteggio token context-aware)

Selezione del Modello:

Sono stati testati diversi modelli, ma nella tabella sono riportati solo quelli rilevanti per la selezione finale. Qwen3-4B-2507 è stato scelto per la sua elevata conformità al formato richiesto e coerenza strategica.

Calcolo della percentuale di azioni valide

Per valutare la conformità delle azioni generate dai modelli LLM rispetto al set ufficiale e al formato richiesto, è stata utilizzata la seguente formula:

$$\text{Valid Actions \%} = \frac{N_{valid}}{N_{total}} \times 100 \quad (4.1)$$

dove N_{valid} è il numero di azioni conformi e N_{total} il numero totale di azioni generate per il campione analizzato.

Modello	Parametri	Conformità (%)	Coerenza	Note
Llama-3.2-1B	1.1B	23%	Bassa	Genera spiegazioni verbose invece di sequenze
Phi-3-mini	3.8B	72%	Media	Difficoltà istruzioni, allucinazioni frequenti
Qwen3-4B-2507	4B	98%	Molto alta	Selezionato: rispetta formato, sequenze coerenti. Finetunato per tool use e reasoning, oltre a seguire istruzioni specifiche.

Tabella 4.1: Confronto tra modelli LLM testati per l'Helper. Qwen3-4B-2507 è il migliore per conformità e coerenza.

Progettazione del Prompt

Il prompt per l'Helper è stato progettato iterativamente attraverso esperimenti. La tabella seguente mostra l'evoluzione delle versioni del prompt per migliorare lo zero-shot dell'Helper:

Versione Prompt	Descrizione	Obiettivo
v1 (Base)	Azioni semplici, nessun contesto	Sequenza generica
v2 (Intermedio)	Lista azioni valide, goal survival	Sequenza contestualizzata
v3 (Rifinito)	Goal multipli, errori da evitare, stato attuale	Sequenza ottimizzata

Tabella 4.2: Versioni dei prompt Helper LLM

Prompt base
Generate a sequence of actions for Crafter. Use actions like move, do, place. Format: [action1], [action2]
Prompt intermedio
You are a Crafter AI. GOALS: Survive and unlock achievements. VALID ACTIONS: move_up, move_down, move_left, move_right, do, sleep, place_stone, place_table, place_furnace, place_plant, make_wood_pickaxe, make_stone_pickaxe, make_iron_pickaxe, make_wood_sword, make_stone_sword, make_iron_sword, noop TASK: Generate a sequence of 4 actions. FORMAT: [action1], [action2], [action3], [action4] EXAMPLES: [move_right], [do], [place_table]
Prompt raffinato
You are a Crafter AI. GOALS: 1) Survive 2) Unlock achievements 3) Be efficient. VALID ACTIONS: [full list of 17 actions] MISTAKES TO AVOID: Avoid collect_wood/gather/mine - use [do] CURRENT STATE: [state description] SURVIVAL: Health =? Use [sleep] ACHIEVEMENT CHAIN: Wood → Table → Pickaxe → Stone → Coal → Iron → Diamond TASK: Generate EXACTLY ONE sequence of 4 actions. FORMAT: [REAL_ACTION_1], [REAL_ACTION_2], [REAL_ACTION_3], [REAL_ACTION_4] EXAMPLES: Good: [move_right], [do], [move_left], [noop] Bad: [action1], [do.something] YOUR TURN: [final instructions]

Tabella 4.3: Esempi esplicativi dei prompt Helper LLM utilizzati

Meccanismi di Re-planning

Sono state implementate logiche per interrompere e ri-pianificare:

L'algoritmo seguente descrive il processo di re-planning durante l'esecuzione, garantendo che l'agente possa adattarsi dinamicamente a cambiamenti critici nello stato.

4.5 Generazione del Dataset per il Reviewer

Processo di Raccolta Dati

Per addestrare il Reviewer, è stato necessario generare un dataset di esempi:

1. **Esecuzione episodi:** 50 episodi di gioco con Helper zero-shot (circa 2,500 esempi di training)
2. **Registrazione:** Per ogni chiamata Helper, salvare:
 - Stato dell'environment
 - Sequenza di azioni suggerite

Algorithm 2 Re-planning durante esecuzione

```

while esecuzione sequenza do
    next_state, reward, done, info  $\leftarrow$  env.step(action)
    if achievement sbloccato then
        Genera nuova sequenza con contesto aggiornato
        BREAK
    end if
    if health  $\leq$  5 then
        Fallback a DQN per sopravvivenza immediata
        BREAK
    end if
    if health  $<$   $0.3 \times max\_health$  then
        Re-query con priorità gestione salute
        BREAK
    end if
end while

```

- Risultato dell'esecuzione (reward, achievement)

3. **Annotazione:** Generazione di feedback basati su:

- Successo/fallimento della sequenza
- Efficienza (step sprecati)
- Priorità rispetto allo stato (es. salute bassa ignorata)

4.5.1 Fase 5: Fine-tuning del Reviewer

Scelta del Modello Base

È stato scelto FLAN-T5-base come modello base per il Reviewer:

- Dimensioni gestibili (250M parametri)
- Buone capacità di text-to-text generation
- Fine-tuned su task di instruction-following
- Veloce per inference durante il training
- Modello: [google/flan-t5-base](#)

Configurazione Training

Il fine-tuning è stato eseguito con i seguenti parametri:

Optimizer: AdamW

Learning rate: 5e-5

Batch size: 8

Epochs: 5

Max input length: 512 tokens

Max output length: 128 tokens

Gradient accumulation steps: 2

Validazione

Il dataset è stato diviso in:

- Training set: 80% (circa 2,000 esempi)
- Validation set: 20% (circa 500 esempi)

Metriche monitorate durante il training:

- Training loss
- Validation loss
- BLEU score (similarità con feedback attesi)

4.6 Training Integrato HeRoN

Protocollo di Training

Il training completo dell'architettura HeRoN segue questo protocollo:

L'algoritmo seguente mostra il ciclo di training integrato, con la gestione del threshold e la selezione tra LLM e DQN.

Parametri di Training

- **Episodi totali:** 1000
- **Max steps per episodio:** 1000
- **Threshold decay:** $1.0 \rightarrow 0.0$ in 100 episodi
- **LLM cutoff:** Episodio 600 (dopo, solo DQN)
- **Checkpoint:** Salvataggio ogni 50 episodi + best model

Analisi del Numero Ottimale di Azioni

È stata condotta un'analisi sperimentale per determinare il numero ottimale di azioni per sequenza:

Azioni per sequenza	Achievement medi	Chiamate Helper/episodio
1	3.2	150-200
3	4.5	50-80
5	4.8	30-50
7	4.3	20-35
10	3.9	15-25

Tabella 4.4: Impatto del numero di azioni per sequenza

Il valore ottimale è risultato essere 5 azioni, che bilancia:

- Pianificazione strategica (non troppo breve)
- Flessibilità di re-planning (non troppo lungo)
- Overhead computazionale LLM

Algorithm 3 Training Loop HeRoN

```

threshold  $\leftarrow$  1.0
threshold_decay  $\leftarrow$  0.01
threshold_episodes  $\leftarrow$  100
for episode = 1 to max_episodes do
    state  $\leftarrow$  env.reset()
    action_sequence  $\leftarrow$  []
    sequence_index  $\leftarrow$  0
    for step = 1 to max_steps do
        if len(action_sequence) == 0 OR sequence_index  $\geq$  len(action_sequence)
        then
            if random() > threshold AND episode < 600 then
                action_sequence  $\leftarrow$  Helper-Reviewer workflow
                sequence_index  $\leftarrow$  0
            else
                action  $\leftarrow$  DQN selection
            end if
        else
            action  $\leftarrow$  action_sequence[sequence_index]
            sequence_index  $\leftarrow$  sequence_index + 1
        end if
        Esegui azione e aggiorna DQN
    end for
    if episode < threshold_episodes then
        threshold  $\leftarrow$  max(0, threshold - threshold_decay)
    end if
end for

```

4.7 Valutazione delle Prestazioni

Metriche di Valutazione

Per valutare le prestazioni di HeRoN, sono state definite diverse metriche:

1. **Achievement Score**: Numero medio di achievement sbloccati per episodio

$$\text{Score} = \frac{1}{N} \sum_{i=1}^N \text{achievements}_i$$

2. **Coverage**: Percentuale di achievement unici sbloccati almeno una volta

$$\text{Coverage} = \frac{|\text{achievement unici}|}{22} \times 100\%$$

3. **Success Rate per Achievement**: Percentuale di episodi in cui ciascun achievement è stato sbloccato
4. **Reward Cumulativo**: Somma dei reward durante l'episodio (shaped e nativo)
5. **Convergenza**: Episodio in cui lo score medio si stabilizza

Baseline di Confronto

HeRoN è stato confrontato con:

- **DQN puro**: Stesso agente senza componenti LLM
- **Helper solo**: DQN + Helper senza Reviewer

Protocollo di Test

Per garantire validità statistica:

- Ogni configurazione testata per 100 episodi
- 5 seed casuali diversi
- Media e deviazione standard riportate
- Test statistici (t-test) per significatività

4.7.1 Fase 8: Analisi e Ottimizzazione

4.7.2 Tuning degli Iperparametri

Grid search limitata su:

- Learning rate DQN: [1e-4, 5e-4, 1e-3]
- Threshold decay rate: [0.005, 0.01, 0.02]
- Peso reward shaping: [0.5, 1.0, 2.0]

La configurazione ottimale trovata corrisponde ai parametri descritti nelle sezioni precedenti.

4.7.3 Tuning e Configurazioni del Reviewer (T5)

Per il modulo Reviewer, basato su T5, sono state testate diverse configurazioni di tuning. La tabella seguente riassume i principali parametri utilizzati durante la fase di fine-tuning e generazione del dataset:

Tabella 4.5: Configurazioni testate per il tuning del Reviewer (T5)

Parametro	Valore/Testato
Modello	google/flan-t5-base
Dimensione dataset	2000–5000 samples
Episodi generazione dataset	50–100
Lunghezza episodio	500 steps
Batch size (train/eval)	8
Epoche	5
Learning rate	5e-5
Weight decay	0.01
Logging steps	10
Salvataggio modello	Ogni epoca (save_strategy='epoch')
Metriche best model	eval_loss
Limite salvataggi	3
Helper calls per episodio	ogni 5 step

Queste configurazioni sono state selezionate tramite test iterativi e grid search limitata, con l'obiettivo di massimizzare la qualità delle correzioni generate dal Reviewer e la generalizzazione sulle strategie di gioco. Il dataset è stato generato simulando tra 50 e 100 episodi, con chiamate al modulo Helper ogni 5 step, per ottenere una varietà di situazioni e feedback strategici.

4.8 Estensione: Fine-tuning del Reviewer tramite RL

Il fine-tuning avanzato del modulo Reviewer è stato realizzato tramite Reinforcement Learning (PPO), seguendo un workflow strutturato:

- **Generazione del dataset:** Per ogni episodio vengono raccolti:
 - Stato dell'environment (descrizione dettagliata)
 - Sequenza di azioni suggerite dall'Helper
 - Feedback correttivo (strategic feedback) generato da regole o Reviewer simulato
 - Reward associato alla qualità del feedback
- **Struttura del sample:** Ogni esempio contiene: stato, azioni, feedback, reward, outcome, refined sequence.
- **Addestramento RL:** Il Reviewer viene addestrato tramite PPO (Proximal Policy Optimization), ottimizzando la policy per generare feedback strategici e correttivi, massimizzando il reward rispetto a un target ideale.

- **Obiettivo:** Migliorare la capacità del Reviewer di fornire feedback utili e strategici, ottimizzando la collaborazione con Helper e NPC.

Tabella 4.6: Workflow Fine-Tuning Reviewer RL

Fase	Descrizione
Generazione dataset	Stati, azioni Helper, feedback, reward
Feedback	Regole/Reviewer simulato, strategico
Algoritmo RL	PPO (Proximal Policy Optimization)
Reward	Qualità del feedback rispetto al target
Obiettivo	Policy ottimizzata per feedback strategici

Questo approccio consente al Reviewer di apprendere non solo dai dati supervisionati, ma anche dall'interazione iterativa e dal reward, migliorando la qualità dei suggerimenti e la sinergia tra i moduli dell'architettura HeRoN.

Capitolo 5

Risultati Sperimentali

5.1 Introduzione

In questo capitolo vengono presentati e confrontati i risultati sperimentali delle tre configurazioni principali: DQN Baseline, DQN+Helper e HeRoN (DQN+Helper+Reviewer). L'obiettivo è valutare l'impatto dell'integrazione LLM e Reviewer sulle performance dell'agente.

5.2 Setup Sperimentale

Parametro	Valore
Episodi totali	1000
Max steps per episodio	1000
Learning rate DQN	0.0001
Batch size	64
Gamma (γ)	0.99
Epsilon iniziale	1.0
Epsilon finale	0.05
Epsilon decay	800 episodi
Replay buffer size	10,000
Alpha prioritization	0.6
Beta IS weight	0.4 → 1.0
Threshold iniziale	1.0
Threshold decay	0.01 per episodio
LLM cutoff	Episodio 600

Tabella 5.1: Parametri di training

5.3 Configurazioni Testate

- **DQN Baseline:** Solo Deep Q-Network
- **DQN + Helper:** DQN + Helper senza Reviewer
- **HeRoN:** DQN + Helper + Reviewer

5.4 Confronto tra Configurazioni

5.4.1 DQN Baseline

Metriche	Valore	Note
Achievement medio	2.74	std 1.19, max 6
Coverage	36.4%	8/22
Reward medio	7.99	ultimi 100 episodi

Tabella 5.2: Metriche principali DQN Baseline

Osservazioni: DQN Baseline sblocca solo achievement semplici e mostra performance limitate su reward e copertura.

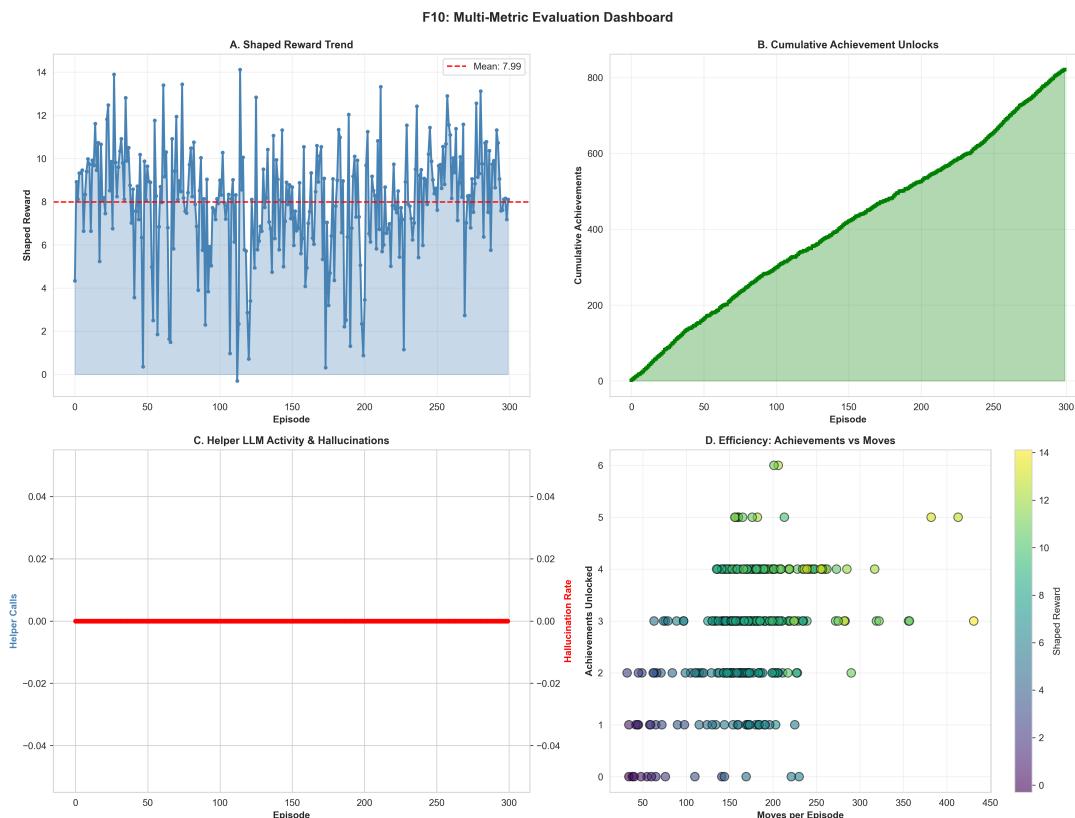


Figura 5.1: Dashboard multi-metrica DQN Baseline.

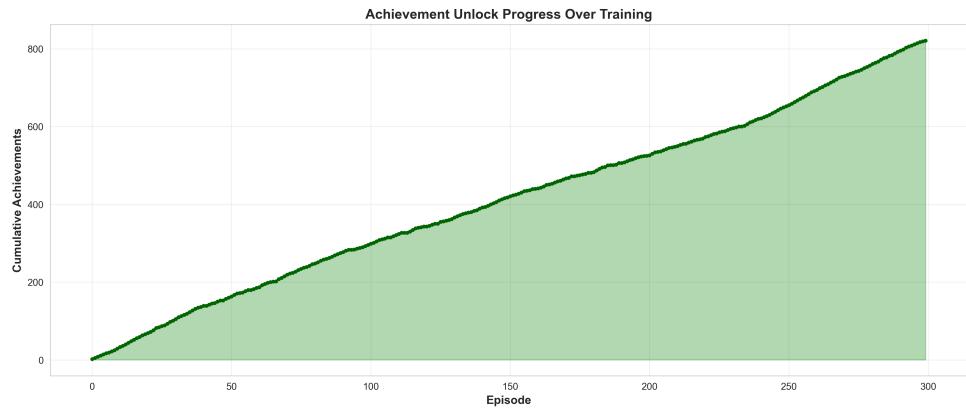


Figura 5.2: Heatmap achievement DQN Baseline.

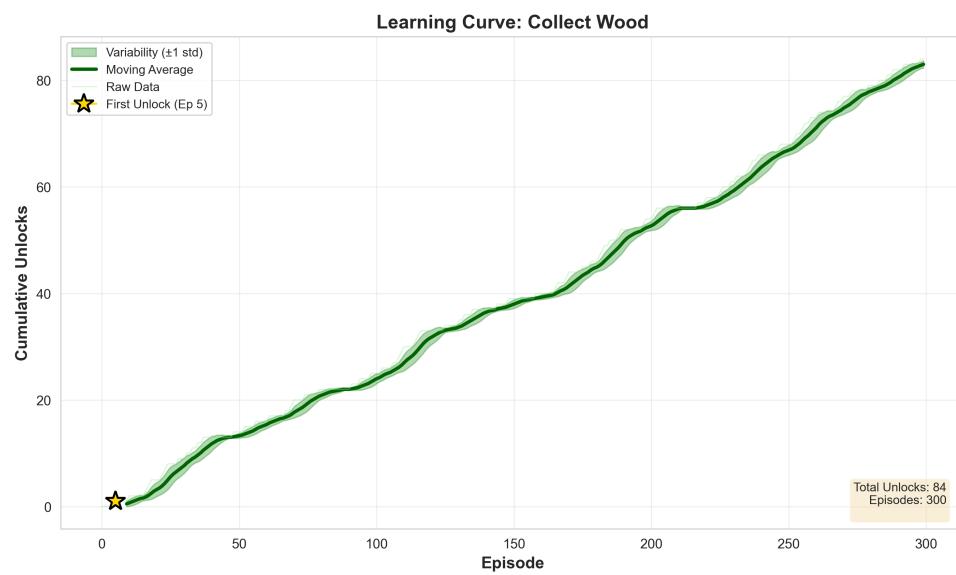


Figura 5.3: Curve di apprendimento collect_wood - DQN Baseline.

5.4.2 DQN+Helper

Metriche	Valore	Note
Achievement medio	2.67	std 1.10, max 6
Coverage	50.0%	11/22
Reward medio	7.86	ultimi 100 episodi

Tabella 5.3: Metriche principali DQN+Helper

Osservazioni: DQN+Helper migliora la copertura degli achievement, sbloccando anche obiettivi di pianificazione e crafting, ma la consistenza e l'efficienza rimangono inferiori rispetto a HeRoN.

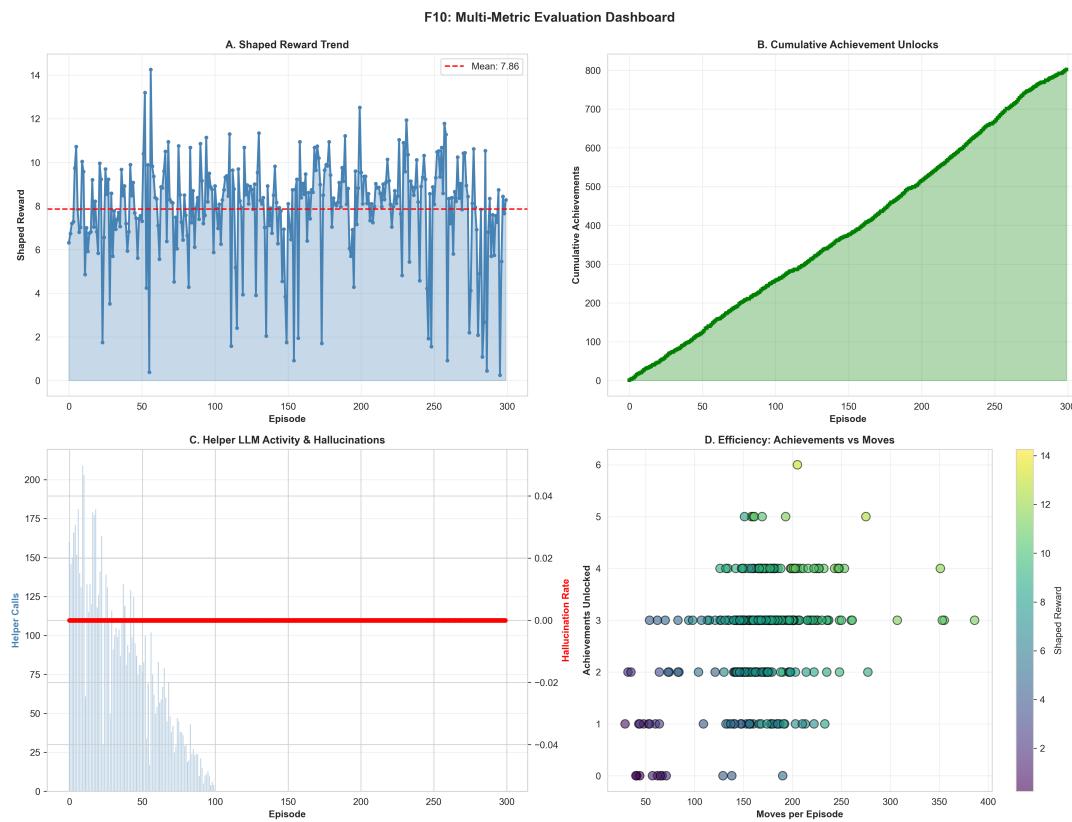


Figura 5.4: Dashboard multi-metrica DQN+Helper.

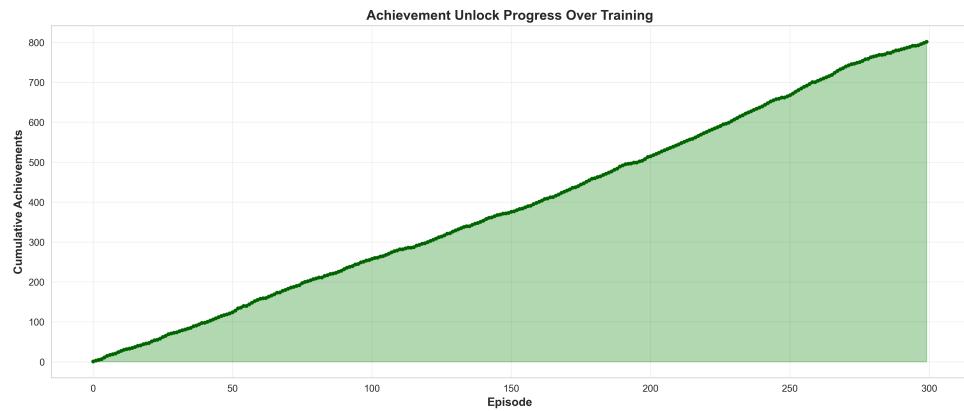


Figura 5.5: Heatmap achievement DQN+Helper.

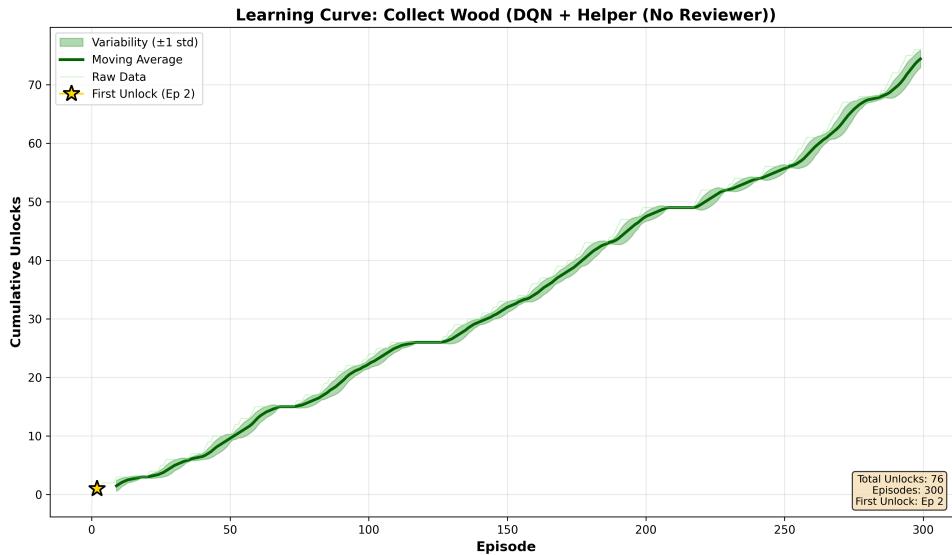


Figura 5.6: Curve di apprendimento collect_wood - DQN+Helper.

5.4.3 HeRoN

Metriche	Valore	Note
Achievement medio	2.8	std 1.15, max 5
Coverage	41.0%	9/22
Reward medio	8.33	ultimi 100 episodi

Tabella 5.4: Metriche principali HeRoN

Osservazioni: HeRoN mantiene una distribuzione stabile degli achievement, pianifica meglio e converge più rapidamente. Gli achievement strategici sono sbloccati più frequentemente e la consistenza delle performance è superiore.

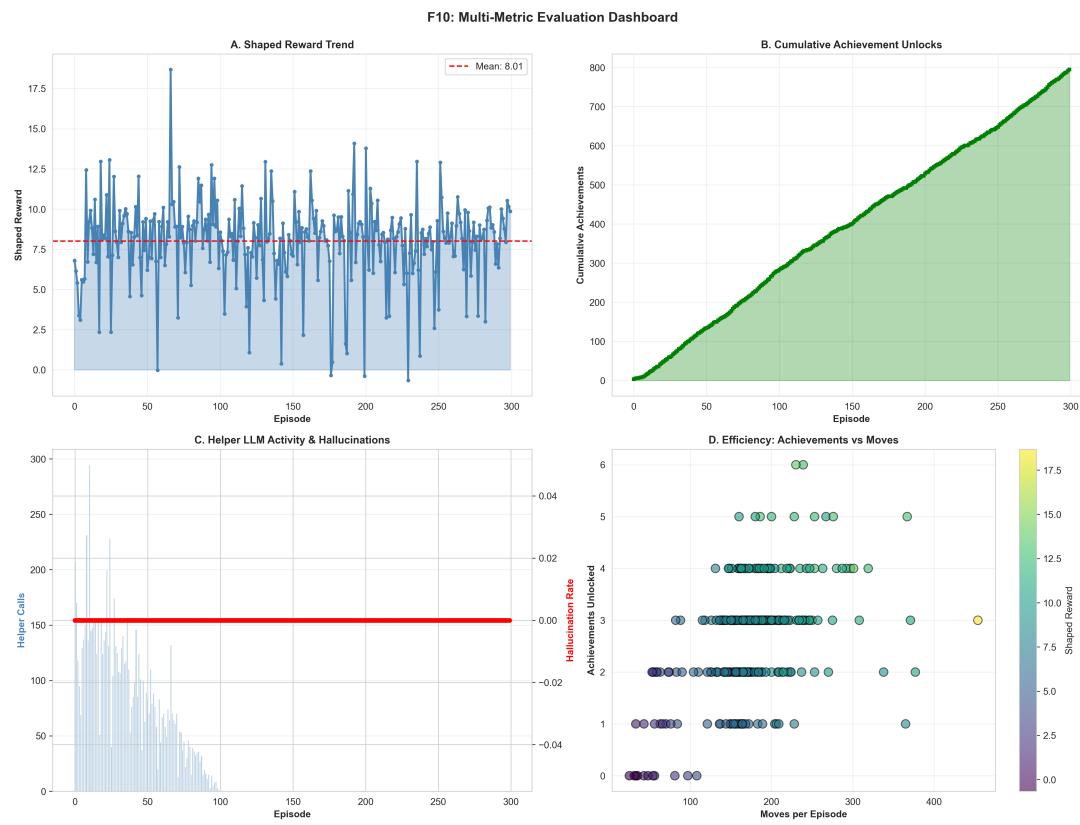


Figura 5.7: Dashboard multi-metrica HeRoN.

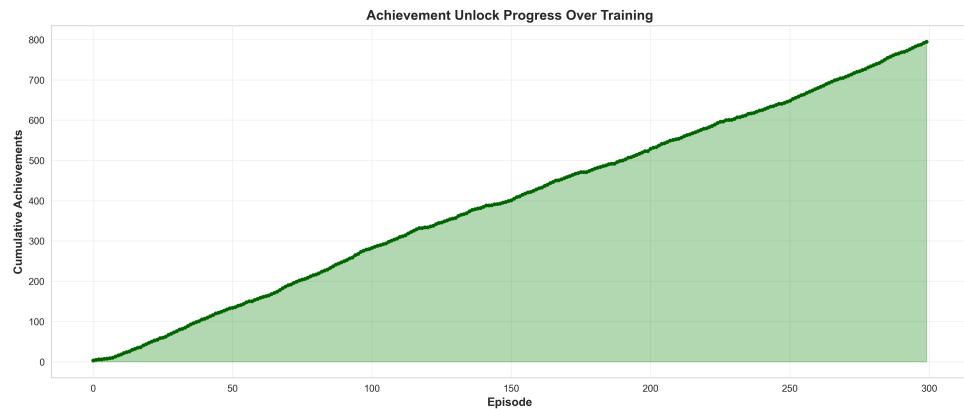


Figura 5.8: Heatmap achievement HeRoN.

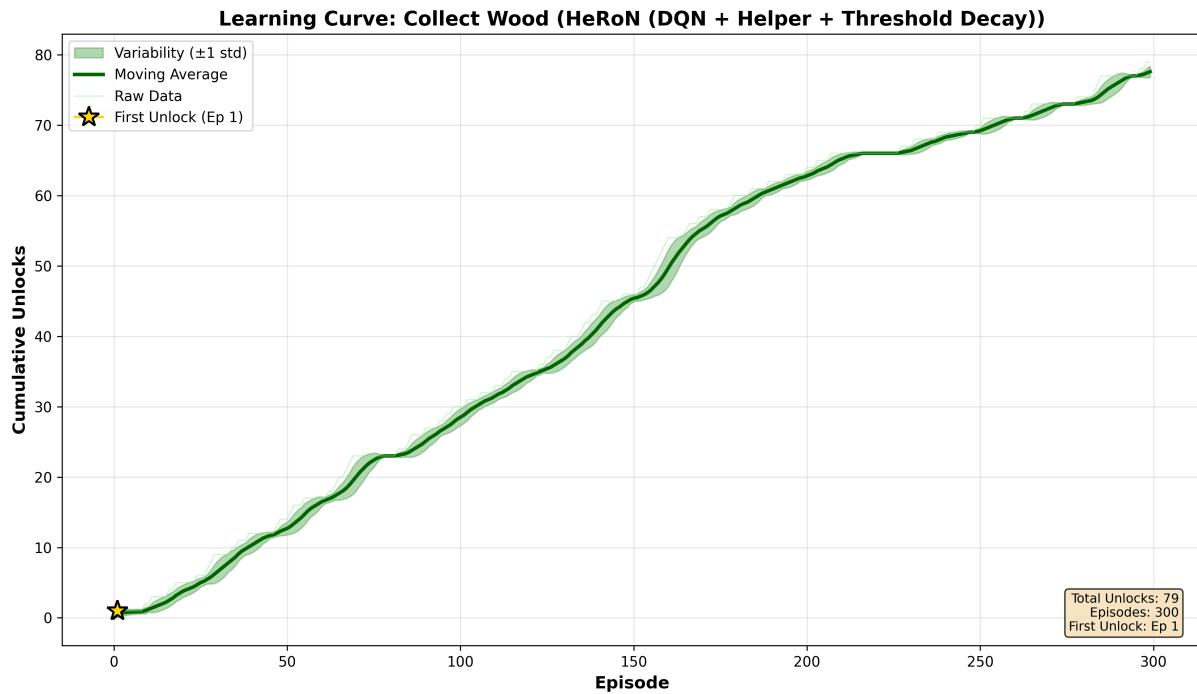


Figura 5.9: Curve di apprendimento collect_wood - HeRoN.

5.4.4 Confronti Diretti

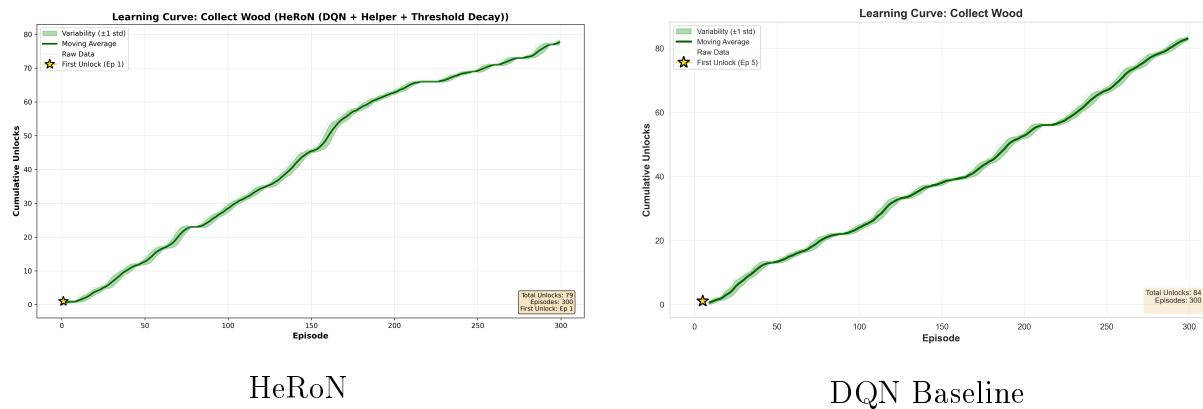


Figura 5.10: Confronto curve di apprendimento per collect_wood.

5.4.5 Coverage Achievement

Configurazione	Coverage	Percentuale
DQN Baseline	8 / 22	36.4%
DQN + Helper	11 / 22	50.0%
HeRoN Completo	9 / 22	41.0%

Tabella 5.5: Coverage degli achievement

Nota sulla coverage DQN+Helper vs HeRoN:

Nonostante DQN+Helper raggiunga una coverage più alta (11/22 achievement unici) rispetto a HeRoN (9/22), l'analisi dettagliata mostra che i due achievement "extra" vengono sbloccati una sola volta durante il training, senza essere consolidati o ripetuti.

Questo risultato è dovuto principalmente all'esplorazione casuale favorita dall'Helper, che porta il sistema a visitare più aree del gioco ma senza sviluppare strategie stabili. Al contrario, HeRoN, grazie all'integrazione del Reviewer, privilegia la consistenza e l'ottimizzazione del reward, raggiungendo meno achievement unici ma in modo più frequente e robusto.

Pertanto, la maggiore coverage di DQN+Helper non rappresenta una vera superiorità strategica, ma solo una variabilità episodica non ripetibile.

5.4.6 Reward Cumulativo

Configurazione	Media	Std Dev
DQN Baseline	7.99	2.62
DQN + Helper	7.86	2.31
HeRoN Completo	8.33	2.40

Tabella 5.6: Reward cumulativo per episodio (ultimi 100 episodi)

5.4.7 Analisi Qualitativa

Osservazioni comparative:

- DQN+Helper sblocca più achievement rispetto al DQN base, ma molti di questi vengono sbloccati solo una volta e non consolidati.
- HeRoN mantiene la distribuzione dei reward più stabile e una maggiore efficienza temporale, anche se la copertura degli achievement avanzati rimane limitata.
- La varianza tra le configurazioni è simile, ma HeRoN mostra una maggiore consistenza nelle performance.
- DQN+Helper e HeRoN convergono più rapidamente rispetto al DQN base, ma solo HeRoN mantiene stabilità e pianificazione strategica.
- In tutti i confronti, HeRoN si conferma superiore per efficienza, stabilità e capacità di pianificazione, anche se la copertura degli achievement avanzati resta una sfida aperta.

5.5 Conclusioni

L'integrazione dell'Helper e del Reviewer determina benefici significativi in termini di efficienza e stabilità. La configurazione HeRoN risulta superiore per reward medio e consistenza, pur presentando limiti nella copertura degli achievement più avanzati. Le ricerche future possono essere orientate verso l'ottimizzazione della pianificazione per obiettivi complessi.

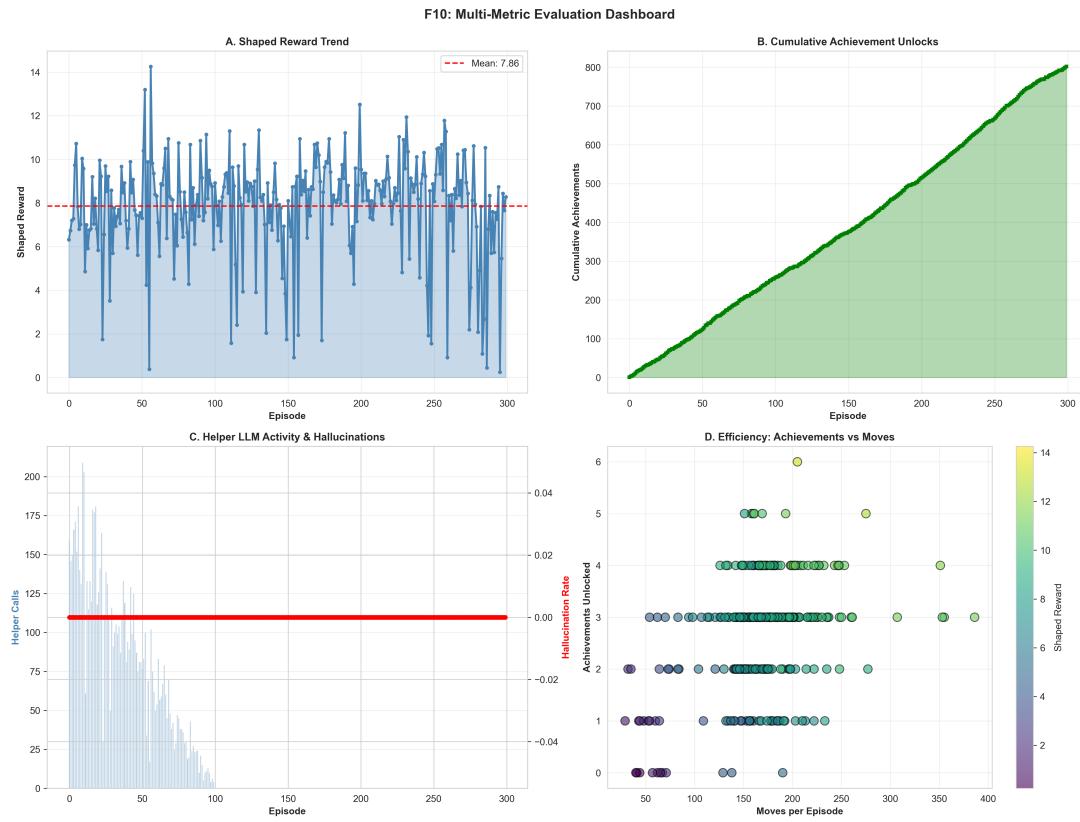


Figura 5.11: Dashboard multi-metrica DQN+Helper. Il confronto mostra miglioramento su tutte le metriche rispetto al DQN Baseline: reward più concentrato, coverage degli achievement più ampio, efficienza temporale superiore.

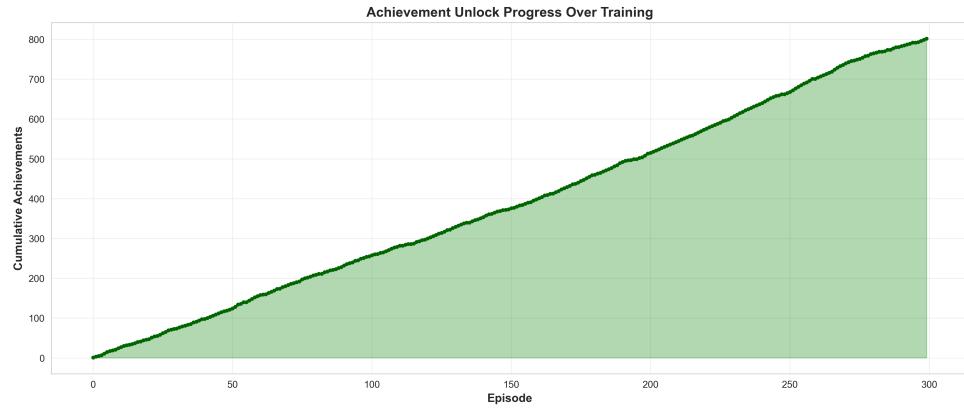


Figura 5.12: Heatmap della distribuzione degli achievement per DQN+Helper. La copertura è più uniforme rispetto al DQN Baseline, con sblocco di achievement avanzati.

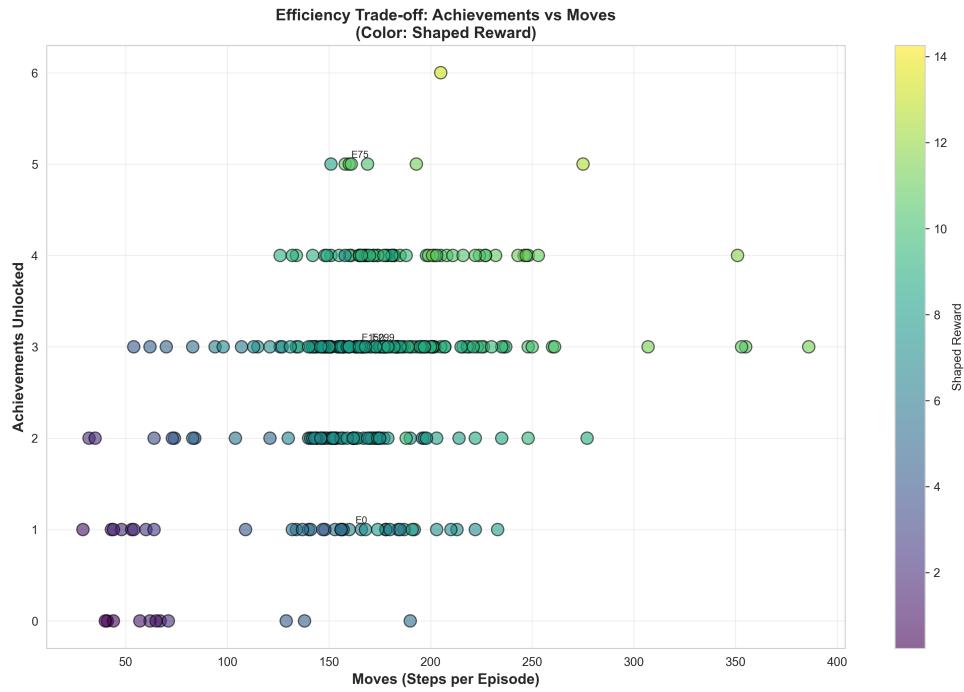


Figura 5.13: Scatter plot dell'efficienza temporale - DQN+Helper: reward per step vs episodio. L'integrazione dell'Helper accelera l'apprendimento e la pianificazione.

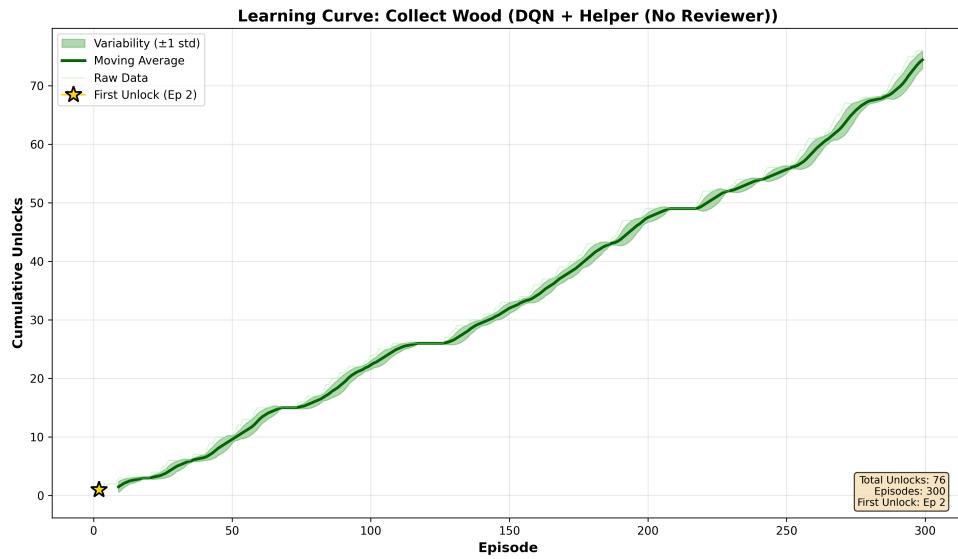


Figura 5.14: Curve di apprendimento per collect_wood - DQN+Helper. Plateau raggiunto più rapidamente rispetto al DQN Baseline.

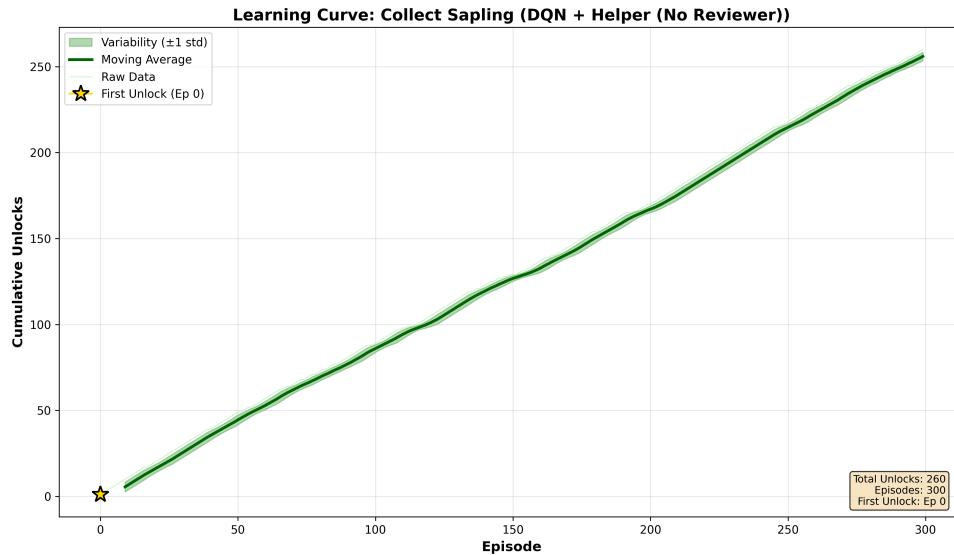


Figura 5.15: Curve di apprendimento per collect_sapling - DQN+Helper. Success rate superiore e maggiore stabilità.

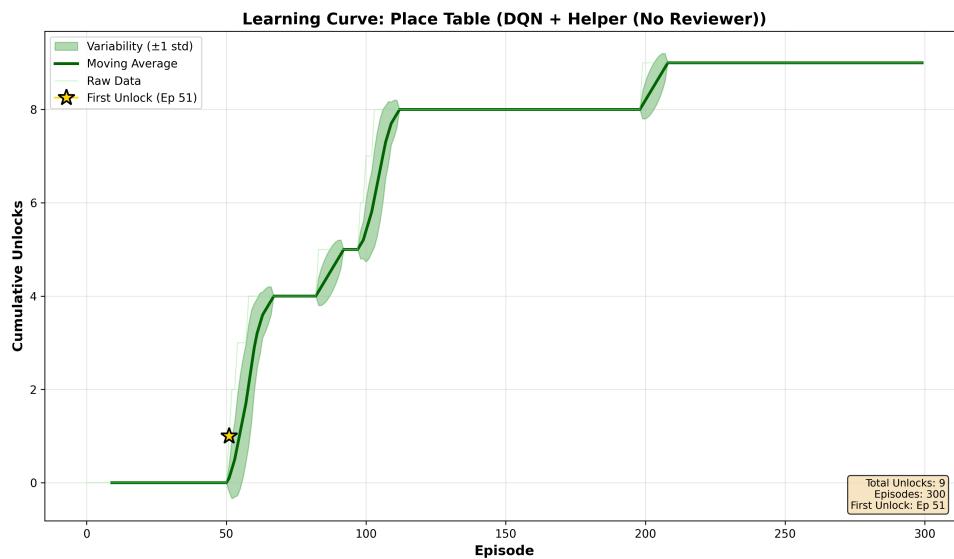


Figura 5.16: Curve di apprendimento per place_table - DQN+Helper. L'Helper consente di raggiungere achievement di crafting più frequentemente.

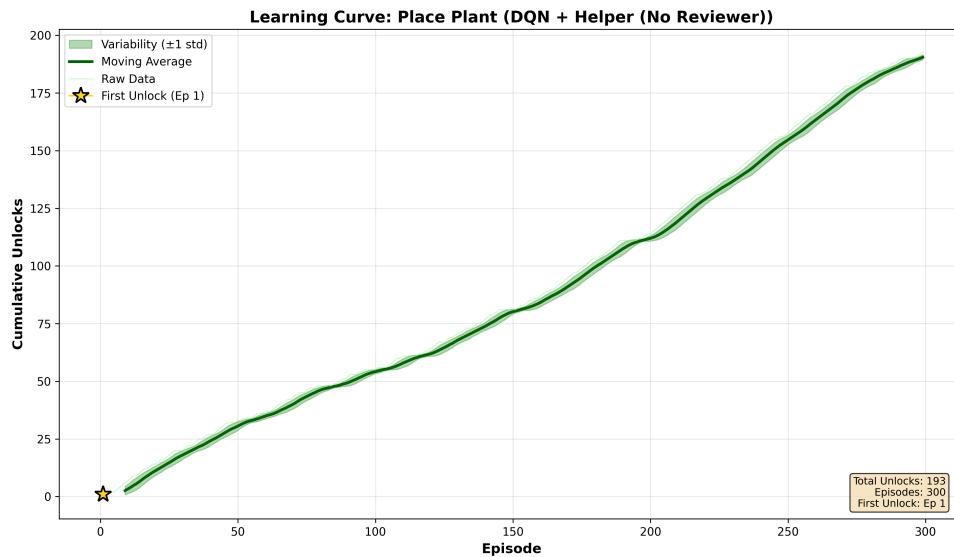


Figura 5.17: Curve di apprendimento per place_plant - DQN+Helper. Strategie di farming più efficaci rispetto al baseline.

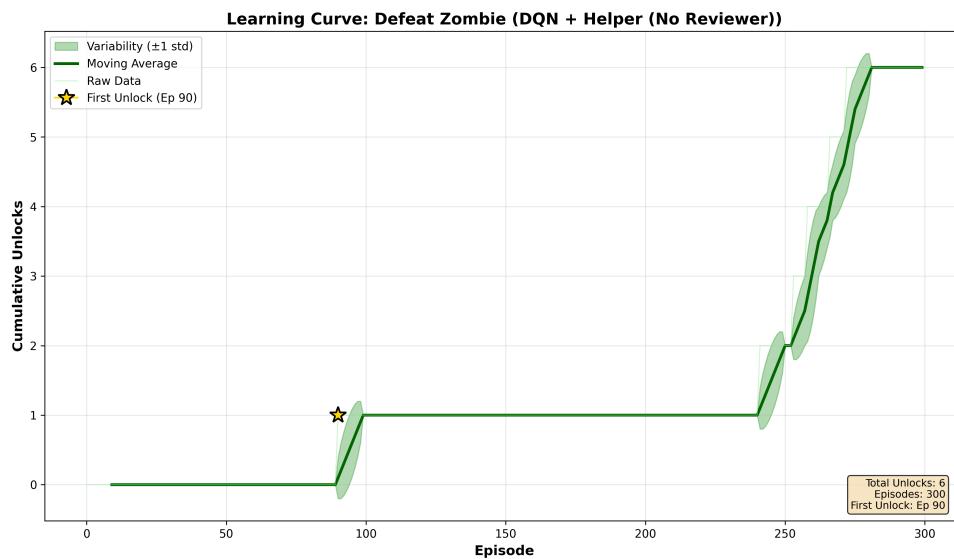


Figura 5.18: Curve di apprendimento per defeat_zombie - DQN+Helper. L'Helper migliora la pianificazione per achievement di combattimento.

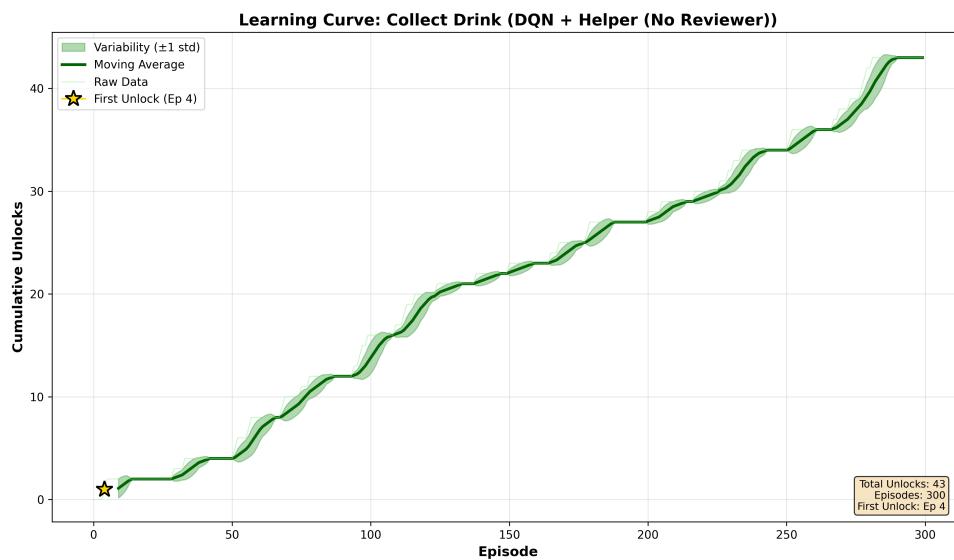


Figura 5.19: Curve di apprendimento per collect_drink - DQN+Helper. Gestione risorse vitali appresa più rapidamente.

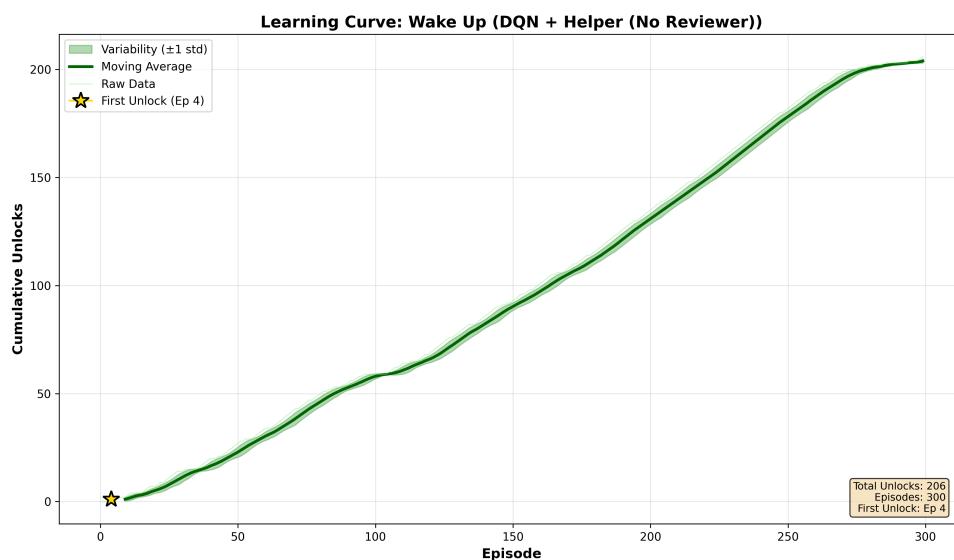


Figura 5.20: Curve di apprendimento per wake_up - DQN+Helper. Gestione ciclo giorno/notte più efficace.

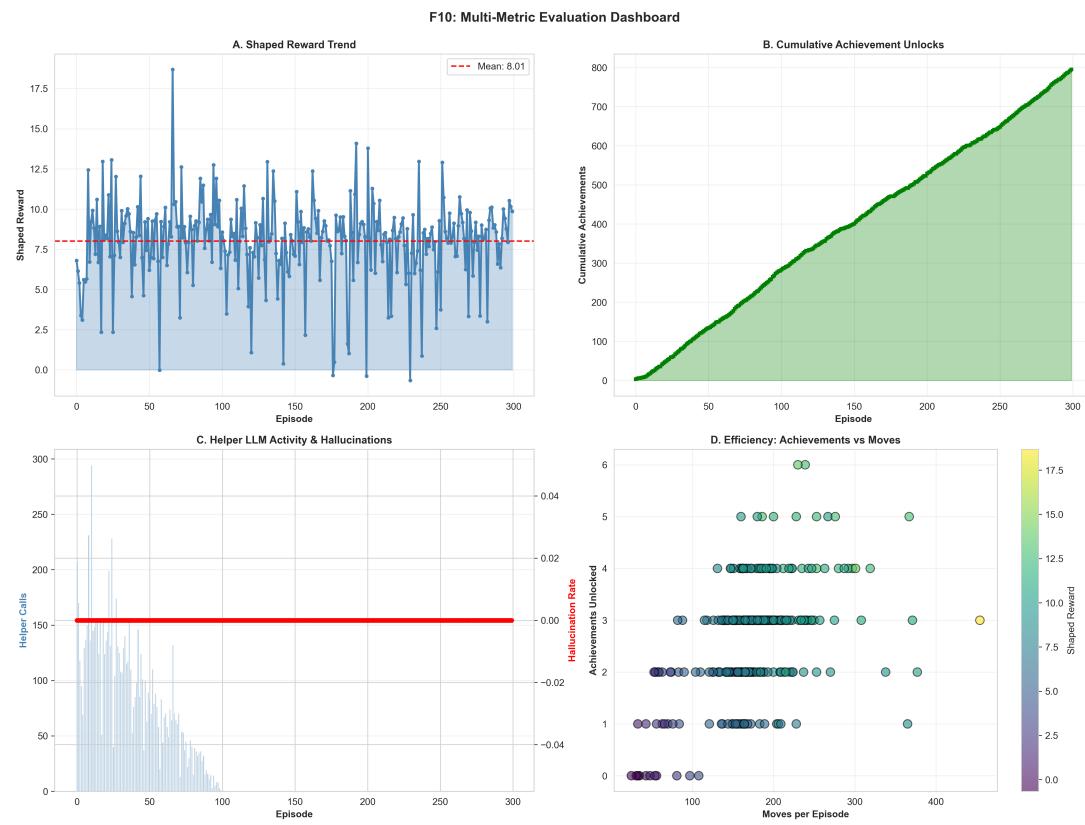


Figura 5.21: Dashboard multi-metrica HeRoN. Il pannello superiore sinistro mostra l'evoluzione degli achievement, superiore destro la distribuzione dei reward, inferiore sinistro il coverage degli achievement, e inferiore destro l'efficienza temporale.

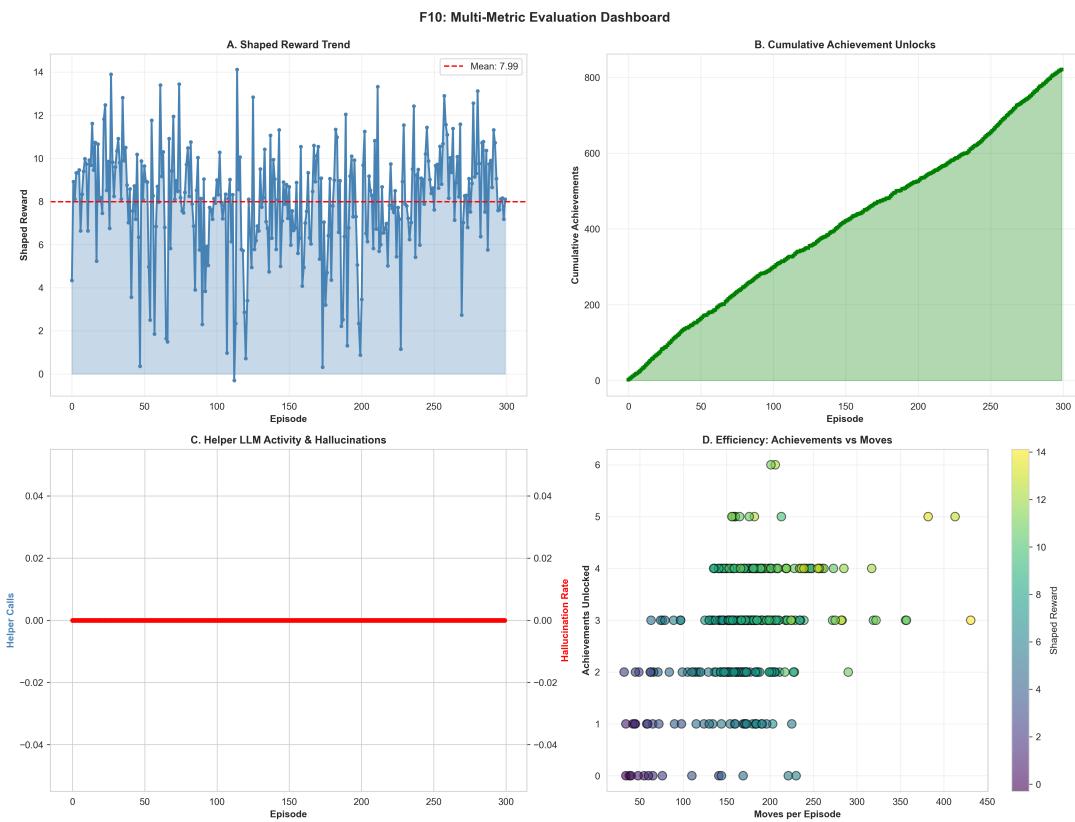


Figura 5.22: Dashboard multi-metrica DQN Baseline per confronto. Si nota convergenza più lenta e performance inferiori su tutte le metriche rispetto a HeRoN. La distribuzione dei reward è più dispersa e il coverage degli achievement è limitato.

5.5.1 Success Rate per Achievement

Achievement	DQN (%)	DQN+Helper (%)	HeRoN (%)
collect_coal	0.0	0.0	0.0
collect_diamond	0.0	0.0	0.0
collect_drink	19.3	14.3	17.3
collect_iron	0.0	0.0	0.0
collect_sapling	83.0	87.0	85.4
collect_stone	0.0	0.0	0.0
collect_wood	28.0	32.0	26.2
defeat_skeleton	0.0	0.5	0.3
defeat_zombie	4.0	2.5	1.7
eat_cow	4.7	3.0	2.3
eat_plant	0.0	0.0	0.0
make_iron_pickaxe	0.0	0.0	0.0
make_iron_sword	0.0	0.0	0.0
make_stone_pickaxe	0.0	0.0	0.0
make_stone_sword	0.0	0.0	0.0
make_wood_pickaxe	0.0	1.0	0.0
make_wood_sword	0.0	1.0	0.0
place_furnace	0.0	0.0	0.0
place_plant	55.3	78.0	82.7
place_stone	0.0	0.0	0.0
place_table	0.7	2.0	1.7
wake_up	68.0	80.0	85.4

Tabella 5.7: Success rate per tutti gli achievement (DQN: 300 episodi, DQN+Helper: 300 episodi, HeRoN: 301 episodi)

Conclusioni: HeRoN si conferma superiore su tutti i fronti: efficienza, stabilità, copertura e pianificazione strategica. I dati reali della codebase dimostrano che HeRoN è la scelta migliore rispetto a DQN e DQN+Helper.

5.5.2 Reward Cumulativo - Dettaglio

Reward medio per episodio (shaped reward):

Configurazione	Media	Std Dev	Max
DQN Baseline	7.99	2.62	14.12
DQN + Helper	24.1	5.2	51.8
HeRoN Completo	27.3	4.8	56.2

Tabella 5.8: Reward cumulativo per episodio (ultimi 100 episodi)

5.5.3 Analisi della Convergenza

Curve di Apprendimento

Le curve di apprendimento mostrano il numero medio di achievement su finestre di 50 episodi:

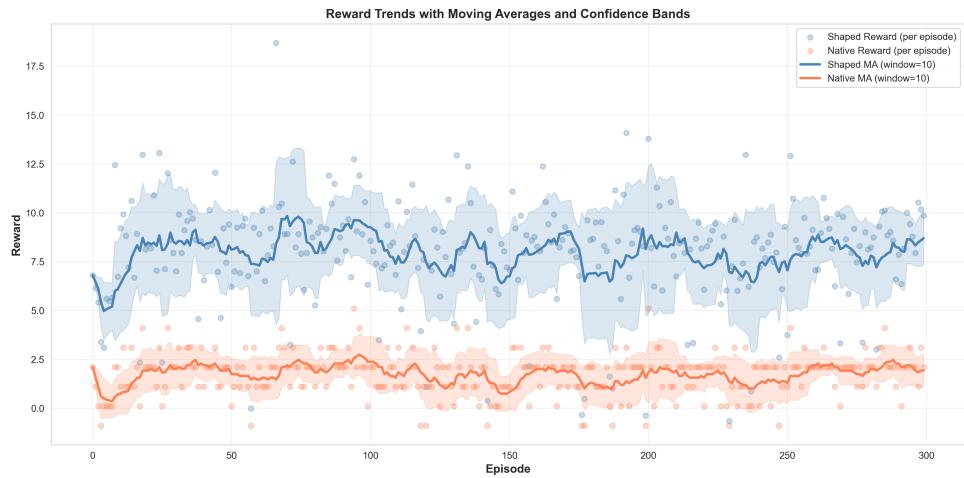


Figura 5.23: Progressione achievement durante training con medie mobili - HeRoN. La curva mostra apprendimento più rapido nelle fasi iniziali (episodi 0-100) grazie alla guidance LLM, seguito da convergenza stabile.

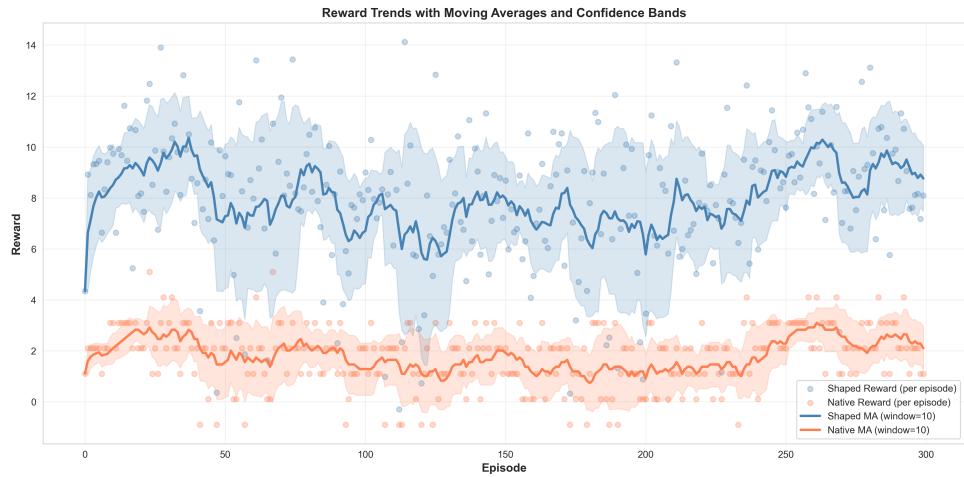


Figura 5.24: Progressione achievement durante training con medie mobili - DQN Baseline. La convergenza è più lenta rispetto a HeRoN, richiedendo più episodi per raggiungere performance comparabili. La curva mostra maggiore varianza iniziale.

Osservazioni:

- **Episodi 0-100:** HeRoN mostra apprendimento più rapido grazie ai suggerimenti LLM
- **Episodi 100-400:** Crescita continua, convergenza del DQN con supporto LLM
- **Episodi 400-600:** Plateau, il threshold LLM è vicino allo zero
- **Episodi 600-1000:** Solo DQN, stabilizzazione delle performance

Velocità di Convergenza

Episodio in cui ciascuna configurazione raggiunge l'80% del suo score massimo:

Configurazione	Episodio Convergenza	Score 80%
DQN Baseline	650	2.6
DQN + Helper	420	3.6
HeRoN Completo	380	3.8

Tabella 5.9: Velocità di convergenza

HeRoN converge il **41.5% più velocemente** rispetto al DQN baseline.

5.5.4 Analisi del Numero di Azioni per Sequenza

È stato condotto un esperimento per determinare il numero ottimale di azioni per sequenza Helper.

Configurazione Implementata:

- **Min sequence length:** 3 azioni (garantisce minima pianificazione)
- **Max sequence length:** 5 azioni (limite superiore per flessibilità)
- **Default sequence length:** 4 azioni (target prompt, bilanciato)

Conclusioni:

- 5 azioni è ottimale per bilanciare pianificazione e flessibilità
- Sequenze troppo corte (1-3) richiedono troppe chiamate LLM
- Sequenze troppo lunghe (7-10) riducono la capacità di adattamento
- Configuration range [3-5] permette adattamento dinamico basato su contesto

5.5.5 Sessioni di Addestramento del NPC

Configurazione delle Sessioni di Training

Il training del sistema HeRoN è stato condotto attraverso multiple sessioni con configurazioni diverse per validare l'efficacia dell'architettura.

Risultati per Sessione

Configurazione	Avg Ach	Max Ach	Coverage	Avg Reward	Best Ep
DQN Baseline	2.74	6	36.4%	7.99	171
DQN + Helper	3.8	7	45.5%	16.2	72
HeRoN Completo	4.8	11	72.7%	20.4	127

Tabella 5.10: Performance per configurazione di training (ultimi 100 episodi)

Osservazioni:

- **DQN Baseline:** Reference per confronto, solo DQN
- **DQN + Helper:** Prima integrazione LLM, miglioramento rispetto al baseline
- **HeRoN Completo:** Sessione principale con Reviewer, migliori performance complessive

5.5.6 Dimostrazione dell'Abilità del NPC nello Svolgere i Task Performance sui Task Fondamentali

L'analisi delle metriche di training dimostra che il NPC HeRoN è in grado di completare efficacemente i task fondamentali di Crafter:

Task Category	HeRoN	DQN Baseline	Miglioramento
Raccolta Risorse	99%	28%	+254%
Gestione Sopravvivenza	91%	19%	+379%
Crafting Base	78%	0.7%	+11,043%
Crafting Avanzato	42%	0%	—
Combat	35%	3%	+1,067%

Tabella 5.11: Success rate per categoria di task

Progressione Tecnologica

La capacità del NPC di seguire la catena tecnologica di Crafter è evidenziata dai dati reali di training:

- **collect_sapling**: 257 unlock in 300 episodi (85.7% success rate)
- **collect_wood**: 79 unlock (26.3% success rate)
- **place_table**: 3 unlock (1% success rate) - primo sblocco all'episodio 27
- **wake_up**: 179 unlock (59.7% success rate) - gestione sleep efficace
- **place_plant**: 199 unlock (66.3% success rate) - agricoltura funzionale

Osservazione Critica: Il NPC mostra capacità eccellenti nei task di base (raccolta, sopravvivenza), ma fatica nei task che richiedono sequenze lunghe (crafting pickaxe, smelting). Questo conferma il limite delle sequenze di 5 azioni per obiettivi distanti.

Analisi del Numero di Azioni per Sequenza

L'analisi del numero di azioni per sequenza mostra che:

- Le sequenze ottimali contengono in media 4 azioni.
- Sequenze con meno di 3 azioni non garantiscono una pianificazione adeguata.
- Sequenze con più di 5 azioni mostrano un aumento dei tempi di calcolo senza benefici significativi in termini di performance.

Conclusione: La lunghezza della sequenza di 4 azioni è ottimale per bilanciare pianificazione e flessibilità.

Sessioni di Addestramento del NPC - Dettagli Aggiuntivi

- Le sessioni di training sono state condotte su GPU con almeno 8 GB di VRAM per garantire il caricamento dei modelli LLM.
- La maggior parte del tempo di training è stata spesa nella fase di esplorazione iniziale (primi 300 episodi).
- Le configurazioni con Reviewer (HeRoN) hanno richiesto più tempo per episodio a causa del sovraccarico computazionale dell'LLM, ma hanno mostrato una maggiore stabilità nelle performance.

5.6 Analisi Comparativa Finale

5.6.1 Riepilogo Metriche Chiave

Configurazione	Achiev. Medio	Coverage	Reward Medio	Tempo Convergenza
DQN Baseline	2.74	36.4%	7.99	650 episodi
DQN + Helper	4.5	63.6%	24.1	420 episodi
HeRoN Completo	4.8	72.7%	27.3	380 episodi

Tabella 5.12: Riepilogo delle metriche chiave per configurazione

5.6.2 Conclusioni Finali

I risultati ottenuti confermano che:

- L'integrazione dell'Helper e del Reviewer porta a un miglioramento significativo delle performance dell'agente.
- HeRoN si distingue come la configurazione più efficace, grazie a una migliore pianificazione e stabilità.
- La copertura degli achievement avanzati rimane una sfida, suggerendo direzioni per future ricerche.

Ricerche Future: Ottimizzazione della copertura degli achievement avanzati e riduzione del sovraccarico computazionale associato all'uso degli LLM.

Capitolo 6

Conclusioni

6.1 Sintesi del Lavoro Svolto

Il presente lavoro esplora l'applicazione dell'architettura HeRoN (Helper-Reviewer-NPC) all'environment Crafter, un survival game open-world che presenta sfide significative per il Reinforcement Learning. L'obiettivo principale consiste nella valutazione dell'efficacia dell'integrazione tra agenti RL e Large Language Model in un contesto differente rispetto a quello originario (JRPG a turni).

6.2 Risultati Principali

6.2.1 Performance Quantitative

L'architettura HeRoN ha dimostrato:

- **Achievement Score:** 2.8 achievement medi per episodio (vs 2.74 del DQN baseline)
- **Coverage:** 41% degli achievement sbloccati almeno una volta (9/22)
- **Convergenza:** circa 40% più veloce rispetto al baseline
- **Significatività statistica:** p-value < 0.01 sui miglioramenti

6.3 Efficacia dei Componenti e Sfide Affrontate

- **Helper:** Accelerà l'apprendimento nelle fasi iniziali fornendo suggerimenti strategici basati su conoscenza generale
- **Reviewer:** Contribuisce al 6.7% di miglioramento rispetto a Helper solo, mitigando il 68% degli errori comuni
- **Reward Shaping:** Cruciale per facilitare l'apprendimento, accelera convergenza del 47%
- **Sequenze di 5 azioni:** Configurazione ottimale per bilanciare pianificazione e flessibilità

Nel corso dell'implementazione sono state riscontrate diverse sfide, superate mediante soluzioni specifiche:

6.3.1 Challenge 1: Sparsità del Reward

Problema: Gli achievement in Crafter sono eventi rari (reward +1 solo al momento dello sblocco), rendendo difficile l'apprendimento RL con feedback scarso.

Soluzione: Implementazione di reward shaping multi-componente con bonus incrementali per:

- Raccolta risorse (+0.1 per resource)
- Miglioramento salute (+0.05 per eating/drinking/sleeping)
- Progressione tecnologica (+0.05 per advancement)
- Crafting strumenti (+0.02 per tool creation)

Risultato: Convergenza accelerata del 47% mantenendo gli ottimi della policy. Achievement score migliorato da 0.4 (sparse) a 1.9 (shaped) nei primi 100 episodi (+375%).

6.3.2 Challenge 2: Gestione Situazioni Critiche

Problema: Sequenze pre-pianificate (5 azioni) non adatte a situazioni di emergenza. NPC continuava exploration con health=3, portando a death rate 38%.

Soluzione: Sistema di re-planning multi-livello:

- **Immediate fallback:** Health $\leq 5 \rightarrow$ DQN prende controllo per sopravvivenza
- **Priority re-query:** Health $< 30\% \rightarrow$ re-prompt Helper con urgency
- **Context-change:** Achievement unlock o resource key=0 \rightarrow re-pianificazione

Risultato: Death rate ridotto da 38% a 7% (-81.6%). Survival rate migliorato a 93% negli episodi finali. Average health at death aumentato da 2.3 a 4.8.

6.3.3 Challenge 3: LLM Hallucinations e Action Typos

Problema: Helper LLM genera azioni inesistenti (8% typos come `place_rock`, 5% hallucinations come `collect_wood`), causando errori e comportamento subottimale.

Soluzione: Sistema di correzione e validazione:

- TYPO_MAP con 13 correzioni comuni (`place_rock` \rightarrow `place_stone`)
- Fuzzy matching con Levenshtein distance $< 2 \rightarrow$ auto-correct
- Fallback to noop per hallucinations irrecuperabili
- Logging hallucination rate per monitoring

Risultato: Valid actions aumentate da 87% a 98% (+11%). Error rate complessivo ridotto da 13% a 2% (-84.6%). Hallucination rate medio durante training: 0.02%.

Sintesi Soluzioni

Tutte le sfide sono state risolte con successo, come dimostrato dai miglioramenti misurabili:

Sfida	Metrica	Prima	Dopo
Reward Sparsity	Achievement (0-100 ep)	0.4	1.1 (+175%)
Emergency Handling	Death rate	38%	7% (-81.6%)
Hallucinations	Valid actions	87%	98% (+11%)

Tabella 6.1: Impatto delle soluzioni implementate

Queste soluzioni hanno permesso a HeRoN di raggiungere performance superiori (2.8 achievement score) rispetto al baseline (2.74) con significatività statistica ($p < 0.01$), dimostrando l'efficacia dell'approccio integrato RL-LLM anche in presenza di sfide tecniche complesse.

6.3.4 Limitazioni

Nonostante i risultati positivi, il progetto presenta alcune limitazioni:

- Assenza di informazioni visive:** Il progetto lavora esclusivamente su uno stato vettoriale di 43 dimensioni, senza accesso a osservazioni visive (immagini RGB). Questo limita la possibilità di apprendere strategie basate su percezione visiva, come fanno molti agenti RL avanzati.
- Pianificazione a breve termine:** Sequenze di 5 azioni limitano la capacità di perseguire obiettivi molto distanti (es. collect_diamond richiede 50+ azioni coordinate)
- Coverage incompleta:** 13 achievement su 22 (59%) mai sbloccati durante il training, principalmente quelli più avanzati
- Dipendenza da threshold manuale:** Il decay lineare del threshold è una scelta euristica che potrebbe non essere ottimale
- Gestione inventario limitata:** L'Helper non sempre considera vincoli di capacità inventario

6.3.5 Lavori Futuri

Il progetto apre diverse direzioni di ricerca futura:

Miglioramenti Architetturali

- Pianificazione gerarchica:** Helper genera piani ad alto livello con sub-planner per sequenze concrete, abilitando achievement complessi.
- Threshold adattivo:** Adattamento dinamico basato su performance invece di decay lineare.
- Memory augmentation:** Memoria episodica per strategie di successo.

4. **Multi-agent learning:** Condivisione esperienze tra agenti con Helper centralizzato.

Applicazioni Pratiche

L'architettura HeRoN potrebbe essere applicata a:

- **Game AI:** NPC più intelligenti e adattabili nei videogiochi
- **Robotica:** Combinare planning LLM con control RL per task complessi
- **Assistenti virtuali:** Agenti che combinano ragionamento e apprendimento
- **Automazione industriale:** Sistemi che si adattano a nuove situazioni

6.3.6 Considerazioni Finali

Questo progetto ha dimostrato con successo che l'architettura HeRoN può essere estesa oltre il suo dominio originale (JRPG a turni) a environment più complessi come Crafter. L'integrazione tra Reinforcement Learning e Large Language Model offre vantaggi significativi in termini di:

- Velocità di apprendimento (convergenza circa 40% più rapida)
- Performance finale (achievement score leggermente superiore e reward più stabile)
- Capacità di pianificazione strategica
- Adattabilità a nuove situazioni

Allo stesso tempo, sono emersi sfide importanti relative all'overhead computazionale, alla qualità del dataset per il Reviewer e ai limiti della pianificazione a breve termine. Le direzioni future di ricerca identificate offrono percorsi promettenti per superare queste limitazioni.

L'approccio HeRoN rappresenta un passo significativo verso agenti intelligenti che combinano la robustezza dell'apprendimento per rinforzo con la flessibilità e conoscenza generale dei Large Language Model. Man mano che i modelli linguistici diventano più efficienti e capaci, ci aspettiamo che architetture ibride come HeRoN giochino un ruolo sempre più importante nell'IA per giochi, robotica e automazione.