

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

Questo lavoro presenta l'adattamento e la validazione dell'architettura HeRoN (Helper-Reviewer-NPC) all'environment Crafter, un survival game open-world che rappresenta un benchmark significativo per il Reinforcement Learning. L'architettura integra tre componenti principali: un agente Deep Q-Network (DQN) con Double DQN e Prioritized Experience Replay, un Helper basato su Large Language Model (Qwen3-4B-2507) che genera sequenze strategiche di 3-5 azioni, e un Reviewer fine-tuned (FLAN-T5-base) che fornisce feedback correttivi per migliorare i suggerimenti dell'Helper.

L'implementazione affronta sfide tecniche complesse: la sparsità del reward nativo di Crafter viene mitigata attraverso un sistema di reward shaping multi-componente; la gestione delle situazioni critiche è garantita da meccanismi di re-planning che interrompono le sequenze pre-pianificate in caso di emergenza; le allucinazioni dell'LLM sono corrette mediante un sistema di validazione e fuzzy matching.

Lo spazio di stati è rappresentato da un vettore a 43 dimensioni comprendente inventario (16 item), posizione, statistiche vitali e achievement sbloccati (22 obiettivi). Il training integrato utilizza un meccanismo di threshold decay che riduce progressivamente l'intervento LLM da 100% a 0% nei primi 100 episodi, permettendo all'agente DQN di acquisire gradualmente autonomia.

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	5
1.1 Contesto	5
1.2 Motivazione e Obiettivi	5
1.2.1 Obiettivi Primari	5
1.2.2 Obiettivi Secondari	6
2 Architettura HeRoN	7
2.1 Panoramica dell'Architettura	7
2.1.1 Diagramma Architettura DQN Baseline	7
2.1.2 Diagramma Architettura HeRoN Completa	7
2.1.3 NPC (Non-Player Character) - Versione Semplificata	8
2.1.4 Helper	9
2.1.5 Reviewer	10
2.1.6 Gestione del Contesto	10
2.1.7 Gestione Intelligente del Contesto (Token-Aware)	10
2.1.8 Meccanismi di Re-planning e Aggiornamento Contesto	11
2.1.9 Reset Episodio e Pulizia Contesto	11
2.2 Workflow dell'Architettura	12
2.2.1 Fase 1: Decisione di Consultazione	12
2.2.2 Fase 2: Review e Raffinamento (Reviewer)	12
2.2.3 Fase 3: Esecuzione e Re-planning	12
2.3 Vantaggi dell'Architettura	13
2.4 Sfide dell'Integrazione	13
3 Environment Crafter	14
3.1 Introduzione a Crafter	14
3.1.1 Caratteristiche Principali	14
3.2 Meccaniche di Gioco	14
3.2.1 Obiettivi di Sopravvivenza	14
3.2.2 Sistema di Progressione	15
3.3 Spazio di Stati	15
3.3.1 Spazio delle Azioni	16
3.3.2 Sistema di Reward	16
4 Metodologia di Implementazione	18
4.1 Introduzione	18
4.2 Panoramica del Processo	18
4.3 NPC (DQN Agent)	18
4.3.1 Architettura della Rete Neurale	18

4.4	Helper (LLM) e Prompt Design	21
4.5	Generazione del Dataset per il Reviewer	23
4.5.1	Fase 5: Fine-tuning del Reviewer	24
4.6	Training Integrato HeRoN	25
4.7	Valutazione delle Prestazioni	26
4.7.1	Fase 8: Analisi e Ottimizzazione	27
4.7.2	Tuning degli Iperparametri	27
4.7.3	Tuning e Configurazioni del Reviewer (T5)	27
4.8	Estensione: Fine-tuning del Reviewer tramite RL	28
4.8.1	Reward Function per PPO	28
5	Risultati Sperimentali	30
5.1	Introduzione	30
5.2	Setup Sperimentale	30
5.3	Configurazioni Testate	30
5.4	Confronto tra Configurazioni	31
5.4.1	Tabella Comparativa delle Metriche Principali	31
5.4.2	Dettaglio Achievement per Configurazione	31
5.4.3	DQN Baseline	31
5.4.4	DQN+Helper	32
5.4.5	HeRoN	33
5.4.6	Confronti Visivi	35
5.4.7	Analisi Qualitativa	35
5.5	Reward Cumulativo - Dettaglio	35
5.6	Analisi della Convergenza	36
5.7	Analisi del Numero di Azioni per Sequenza	37
5.8	Dimostrazione dell'Abilità del NPC nello Svolgere i Task	38
5.8.1	Progressione Tecnologica	38
5.9	Analisi Comparativa Finale	38
5.9.1	Riepilogo Metriche Chiave	38
5.9.2	Conclusioni Finali	39
6	Conclusioni	40
6.1	Sintesi del Lavoro Svolto	40
6.2	Risultati Principali	40
6.2.1	Performance Quantitative	40
6.3	Efficacia dei Componenti e Sfide Affrontate	40
6.3.1	Challenge 1: Sparsità del Reward	41
6.3.2	Challenge 2: Gestione Situazioni Critiche	41
6.3.3	Challenge 3: LLM Hallucinations e Action Typos	41
6.3.4	Limitazioni	42
6.3.5	Lavori Futuri	42
6.3.6	Considerazioni Finali	43

Elenco delle figure

5.1	Dashboard multi-metrica DQN Baseline.	32
5.2	Heatmap achievement DQN Baseline.	32
5.3	Dashboard multi-metrica DQN+Helper.	33
5.4	Heatmap achievement DQN+Helper.	33
5.5	Dashboard multi-metrica HeRoN.	34
5.6	Heatmap achievement HeRoN.	34
5.7	Confronto curve di apprendimento per collect_wood.	35
5.8	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.	36
5.9	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.	37

Elenco delle tabelle

4.1	Confronto tra modelli LLM testati per l'Helper. Qwen3-4B-2507 è il migliore per conformità e coerenza	22
4.2	Versioni dei prompt Helper LLM	22
4.3	Esempi esplicativi dei prompt Helper LLM utilizzati	23
4.4	Parametri di fine-tuning del Reviewer	25
4.5	Impatto del numero di azioni per sequenza	26
4.6	Configurazioni testate per il tuning del Reviewer (T5)	27
4.7	Workflow Fine-Tuning Reviewer RL	28
4.8	Parametri PPO per Fine-Tuning Reviewer	29
5.1	Parametri di training	30
5.2	Confronto metriche principali tra le tre configurazioni	31
5.3	Tasso di sblocco degli achievement (%) per configurazione. Sono riportate le percentuali di episodi nei quali ogni achievement è stato sbloccato almeno una volta.	31
5.4	Reward cumulativo per episodio (ultimi 100 episodi)	36
5.5	Velocità di convergenza	37
5.6	Riepilogo delle metriche chiave per configurazione	38
6.1	Sintesi delle soluzioni implementate	42

Capitolo 1

Introduzione

1.1 Contesto

Il presente lavoro rientra nel 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 e Obiettivi

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 efficacia nel migliorare le prestazioni degli agenti RL.

La sfida principale consiste nell'estensione di HeRoN a un contesto molto diverso: il gioco Crafter, un open-world di sopravvivenza che richiede pianificazione a lungo termine, gestione delle risorse e adattamento dinamico.

1.2.1 Obiettivi Primari

Gli obiettivi principali del lavoro sono:

- **Adattamento architettonico:** Estendere HeRoN dall'environment JRPG a turni a Crafter, un survival game open-world in tempo continuo
- **Fine-tuning del Reviewer:** Adattare il componente Reviewer ai task specifici di Crafter, generando un dataset appropriato e addestrando il modello per feedback efficaci nel survival game
- **Generazione di sequenze:** Modificare l'Helper per generare sequenze di 3-5 azioni coerenti anziché singole decisioni, enabling strategic planning

- **Implementazione DQN:** Sviluppare un agente RL basato su Deep Q-Network ottimizzato per le 17 azioni disponibili e spazio di stati a 43 dimensioni
- **Valutazione comparativa:** Valutare HeRoN quantitativamente rispetto a baseline tradizionali, misurando achievement sbloccati nei 22 obiettivi disponibili

1.2.2 Obiettivi Secondari

- Determinare il numero ottimale di azioni per sequenza dell'Helper
- Analizzare l'impatto del reward shaping sulle prestazioni dell'agente
- Implementare meccanismi di re-planning per 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 fondo consiste nell'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:** Ambiente \rightarrow Estrazione dello stato (vettore 43-dim)
2. **Decisione:** Rete DQN \rightarrow Q-values \rightarrow selezione ϵ -greedy
3. **Azione:** Esegui azione a_t , osserva r_t, s_{t+1}
4. **Memorizzazione:** Salva $(s_t, a_t, r_{shaped}, s_{t+1}, done)$ in Prioritized Replay
5. **Apprendimento:** Campiona batch \rightarrow calcola TD-loss \rightarrow aggiorna pesi DQN
6. **Stabilizzazione:** Ogni 100 passi, copia pesi DQN \rightarrow Rete Target

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. utilizzare 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. Ambiente genera stato \rightarrow Estrazione dello stato (43-dim)
2. Threshold check: $\text{random}(0.73) > \text{threshold}(0.65) \rightarrow \text{LLM Path}$
3. Helper riceve descrizione dello stato \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: `[sleep]`, `[move_right]`, `[do]`, `[move_left]`, `[noop]`
6. Action Executor esegue `[sleep]` $\rightarrow (s_1, r_1, info_1)$
7. Salva $(s_0, sleep, r_1, s_1, done)$ 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 aggiorna pesi

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 5.000 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, esegue 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 si implementa 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 circa lo stato corrente del gioco:
 - Inventario del giocatore (16 item)
 - Posizione corrente
 - Statistiche vitali (salute, cibo, acqua)
 - Achievement sbloccati (22 possibili)
- **Prompt Engineering:** Il prompt si presenta come specificatamente 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. Come descritto in dettaglio nel Capitolo 4, il Reviewer si addestra specificamente per il contesto di Crafter su un dataset generato da circa 50 episodi di gioco, contenente circa 2,500 esempi di coppie (suggerimento, feedback).

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 si utilizza per ri-interrogare l'Helper con più informazioni.

2.1.6 Gestione del Contesto

Durante ogni episodio, l'Helper mantiene uno stato conversazionale persistente (`message_history`) che accumula descrizioni dello stato di gioco, sequenze di azioni generate, feedback del Reviewer e contesto di gioco (posizione, inventario, achievement). Questa memoria conversazionale consente all'Helper di mantenere coerenza e contestualizzazione lungo l'episodio, influenzando direttamente la qualità dei suggerimenti generati.

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.

2.1.7 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:** L'Helper conta il numero di token nella cronologia utilizzando 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, l'Helper genera un riassunto 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

2.1.8 Meccanismi di Re-planning e Aggiornamento Contesto

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

Trigger di Re-query (Interruzione Sequenza):

- **Achievement sbloccato:** Quando il giocatore sblocca un nuovo achievement, l'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 l'Helper si consulta per suggerire azioni di emergenza (mangiare, bere, dormire)
- **Salute bassa ($health < 30\%$):** Se la salute è bassa ma non critica, si consulta l'Helper 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, l'Helper aggiorna il suo tracciamento interno registrando i nuovi achievement sbloccati, il numero di passi eseguiti, il reward accumulato e il feedback più recente del Reviewer. Questi dati vengono utilizzati durante il reset del contesto per generare un riassunto episodico coerente, assicurando che il LLM comprenda il progresso complessivo anche dopo un reset di contesto.

2.1.9 Reset Episodio e Pulizia Contesto

All'inizio di ogni nuovo episodio, l'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.

2.2 Workflow dell'Architettura

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

2.2.1 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 < 100 then
    Procedi con workflow LLM
else
    Usa solo DQN:  $a = \arg \max_a Q(s, a)$ 
end if
```

Quando si decide 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, durante le fasi iniziali dell'allenamento si fa ricorso ai suggerimenti LLM, per poi ridurre questa dipendenza man mano che l'agente RL migliora.

2.2.2 Fase 2: Review e Raffinamento (Reviewer)

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

2.2.3 Fase 3: Esecuzione e Re-planning

La sequenza di azioni viene eseguita una dopo l'altra. Durante l'esecuzione, l'Helper monitora determinati eventi trigger che comportano l'interruzione della sequenza corrente e l'attivazione di una nuova query:

- **Achievement sbloccato:** Quando il giocatore sblocca un nuovo achievement, l'Helper riceve una nuova query con il contesto aggiornato
- **Salute critica ($health \leq 5$):** In caso di salute critica, fallback immediato a DQN per azioni di sopravvivenza
- **Salute bassa ($health < 30\%$):** Ri-query del sistema per gestione prioritaria della salute
- **Risorsa chiave consumata:** Se legno, pietra o carbone raggiungono 0, nuova query per raccoglierla prioritariamente

2.3 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

2.4 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 presente lavoro si impiega una rappresentazione strutturata a 43 dimensioni per migliorare efficienza, interpretabilità, apprendimento e compatibilità con LLM:

Inventario (16 dimensioni) Statistiche vitali e conteggio degli oggetti:

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

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:

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

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

$$r_{\text{health}} = 0.02 \quad (\text{se health, food o drink} > 5) \quad (3.3)$$

$$r_{\text{tools}} = 0.3 \times \# \text{ nuovi strumenti crafted} \quad (3.4)$$

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 al fine di orientare la lettura 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

Le fasi principali dello sviluppo sono:

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. Training integrato dell'architettura HeRoN e valutazione

Per una descrizione dettagliata dei componenti architetturali (NPC, Helper, Reviewer) e del meccanismo di threshold decay, si rimanda al Capitolo 2. Le sezioni seguenti sviluppano i dettagli implementativi di ciascuna fase.

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:** 128 neuroni, attivazione ReLU
- **Hidden Layer 2:** 128 neuroni, attivazione ReLU
- **Hidden Layer 3:** 64 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):

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

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 $C = 100$ step tramite hard copy:

$$\theta_{\text{target}} \leftarrow \theta_{\text{policy}}$$

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

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:** 300
- **Max steps per episodio:** 1000
- **Epsilon decay:** $\epsilon = 1.0 \rightarrow 0.05$ in 300 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:

- 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 sottoposto a progettazione iterativa mediante sperimentazione. 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:

Parametro	Valore
Optimizer	AdamW
Learning rate	5e-5
Batch size	8
Epochs	5
Max input length	512 token
Max output length	128 token
Gradient accumulation steps	2

Tabella 4.4: Parametri di fine-tuning del Reviewer

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:

Parametri di Training

- **Episodi totali:** 300
- **Max steps per episodio:** 1000
- **Threshold decay:** $1.0 \rightarrow 0.0$ in 100 episodi
- **LLM cutoff:** Episodio 100 (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 4.5: 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

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 4.6: 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 4.7: 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.

4.8.1 Reward Function per PPO

La reward function utilizzata per l'addestramento PPO del Reviewer è stata progettata per valutare la qualità dei feedback strategici generati. La funzione è multi-componente e combina diversi criteri:

$$r = r_{\text{length}} + r_{\text{terms}} + r_{\text{actions}} + r_{\text{quality}} + r_{\text{penalty}} \quad (4.2)$$

I componenti sono definiti come segue:

- **Penalità lunghezza** (r_{length}): Feedback troppo corti (meno di 10 caratteri) o vuoti ricevono una penalità di -5.0 , poiché non forniscono informazioni utili.
- **Bonus termini strategici** (r_{terms}):

$$r_{\text{terms}} = 0.5 \times \sum_{t \in T} \mathbb{1}[t \in \text{feedback}] \quad (4.3)$$

dove T è l'insieme dei termini strategici specifici di Crafter: *achievement, resource, collect, craft, health, wood, stone, iron, pickaxe, sword, table, prioritize, efficiency, progression, tier*.

- **Overlap azioni** (r_{actions}): Misura la corrispondenza tra le azioni suggerite nel feedback e quelle ideali nel dataset:

$$r_{\text{actions}} = 3.0 \times \frac{|A_{\text{ideal}} \cap A_{\text{suggested}}|}{\max(|A_{\text{ideal}}|, 1)} \quad (4.4)$$

dove A_{ideal} e $A_{\text{suggested}}$ sono rispettivamente gli insiemi di azioni nel feedback target e in quello generato, estratte tramite pattern matching sui tag `[action]`.

- **Indicatori di qualità** (r_{quality}): Un bonus di $+2.0$ viene assegnato se il feedback contiene indicatori strutturati come EXCELLENT, GOOD, CRITICAL, WARNING o SUGGESTION.
- **Penalità verbosità** (r_{penalty}): Feedback eccessivamente lunghi (oltre 500 caratteri) ricevono una penalità di -1.0 per scoraggiare output prolissi e poco concisi.

Pipeline di Addestramento PPO

Il processo di addestramento segue questi passi:

1. **Input**: Il modello riceve la concatenazione di stato del gioco e risposta dell'Helper
2. **Generazione**: Il Reviewer genera un feedback strategico
3. **Calcolo reward**: La reward function valuta la qualità del feedback generato
4. **Aggiornamento policy**: PPO aggiorna i pesi del modello per massimizzare il reward atteso

Tabella 4.8: Parametri PPO per Fine-Tuning Reviewer

Parametro	Valore
Learning rate	5×10^{-7}
PPO epochs	1
Mini batch size	1
Batch size	1
Temperature (generazione)	0.4
Top-k sampling	50
Top-p sampling	0.8
Max new tokens	128

Questa configurazione è stata scelta per garantire stabilità nell'addestramento e generazione di feedback concisi ma informativi.

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 consiste nella valutazione l'impatto dell'integrazione LLM e Reviewer sulle performance dell'agente.

5.2 Setup Sperimentale

Parametro	Valore
Episodi totali	300
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	300 episodi
Replay buffer size	5,000
Alpha prioritization	0.6
Beta IS weight	0.4 → 1.0
Threshold iniziale	1.0
Threshold decay	0.01 per episodio
LLM cutoff	Episodio 100

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 Tabella Comparativa delle Metriche Principali

Metrica	DQN Baseline	DQN+Helper	HeRoN
Achievement medio	2.74 (std 1.19)	2.67 (std 1.10)	2.8 (std 1.15)
Coverage	36.4% (8/22)	50.0% (11/22)	41.0% (9/22)
Reward medio	7.99	7.86	8.33

Tabella 5.2: Confronto metriche principali tra le tre configurazioni

5.4.2 Dettaglio Achievement per Configurazione

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 5.3: Tasso di sblocco degli achievement (%) per configurazione. Sono riportate le percentuali di episodi nei quali ogni achievement è stato sbloccato almeno una volta.

5.4.3 DQN Baseline

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

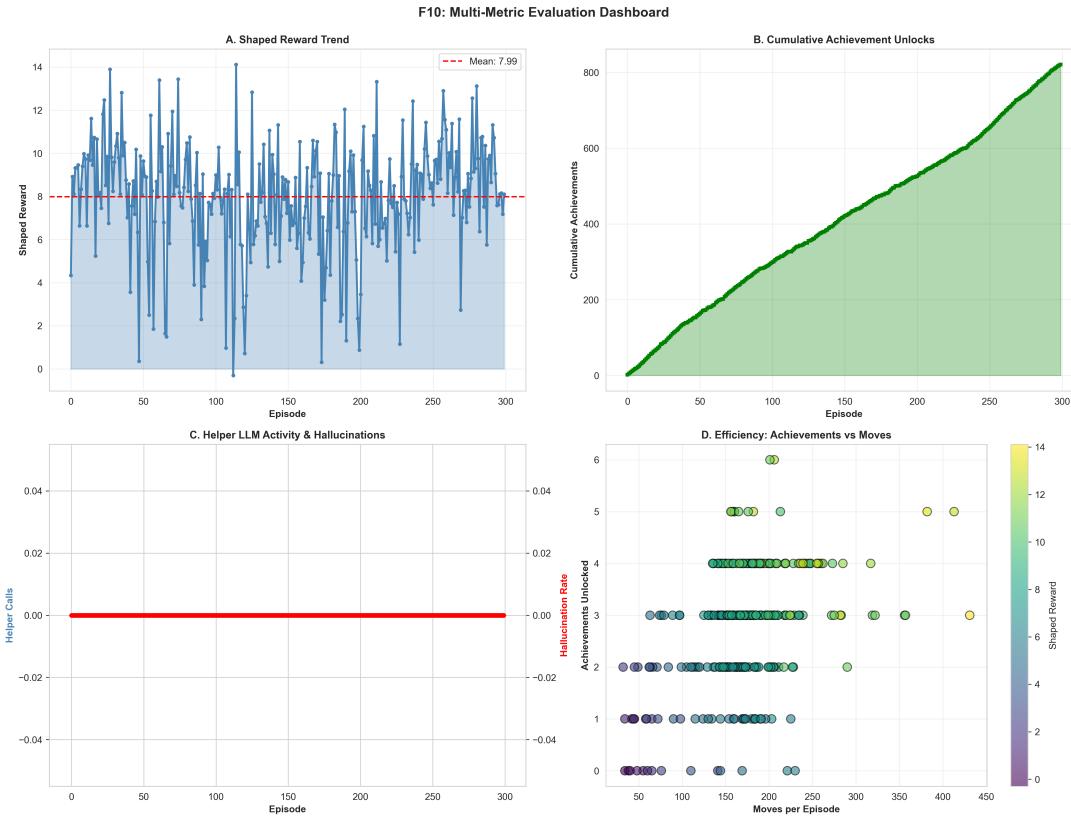


Figura 5.1: Dashboard multi-metrica DQN Baseline.

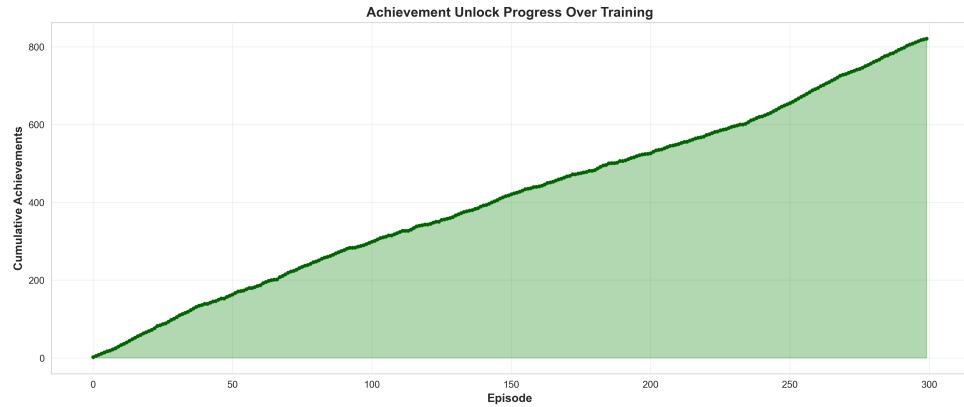


Figura 5.2: Heatmap achievement DQN Baseline.

5.4.4 DQN+Helper

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

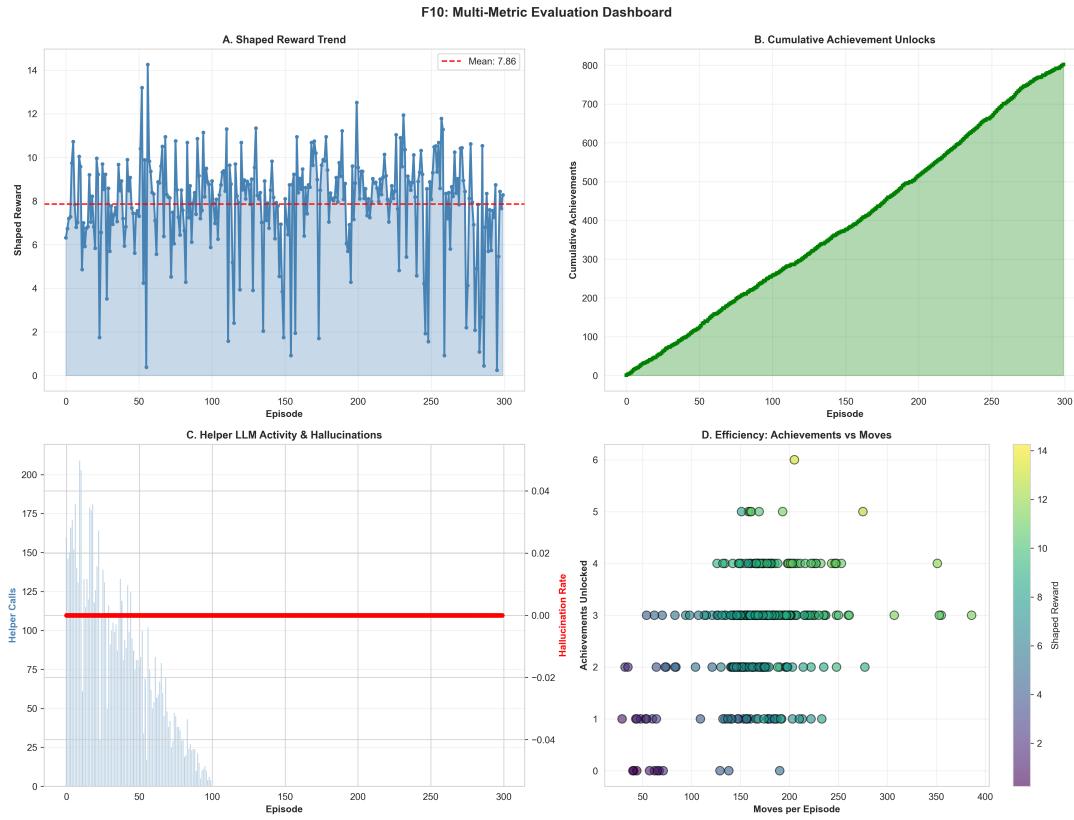


Figura 5.3: Dashboard multi-metrica DQN+Helper.

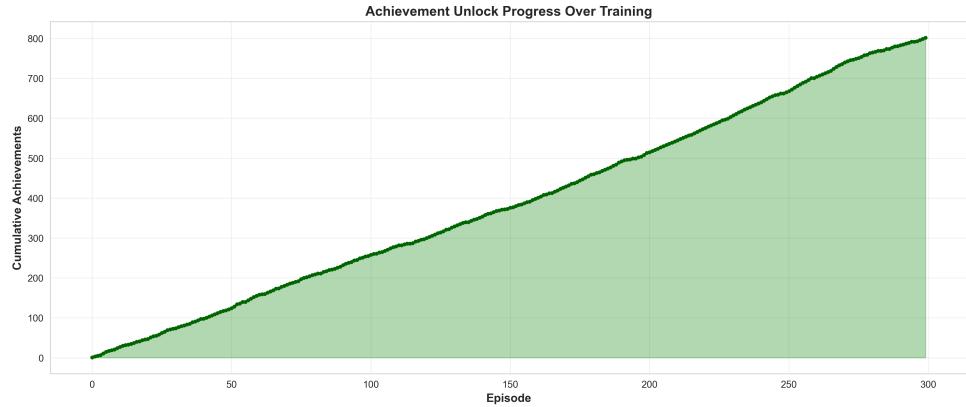


Figura 5.4: Heatmap achievement DQN+Helper.

5.4.5 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.

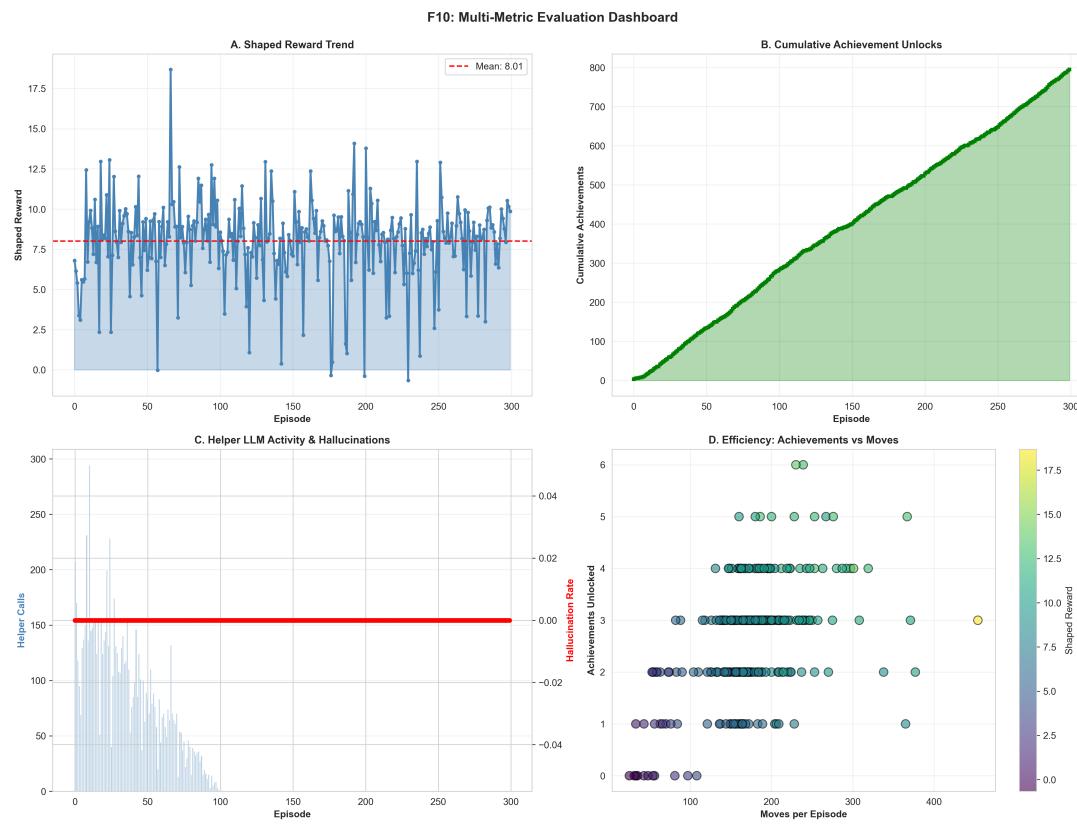


Figura 5.5: Dashboard multi-metrica HeRoN.

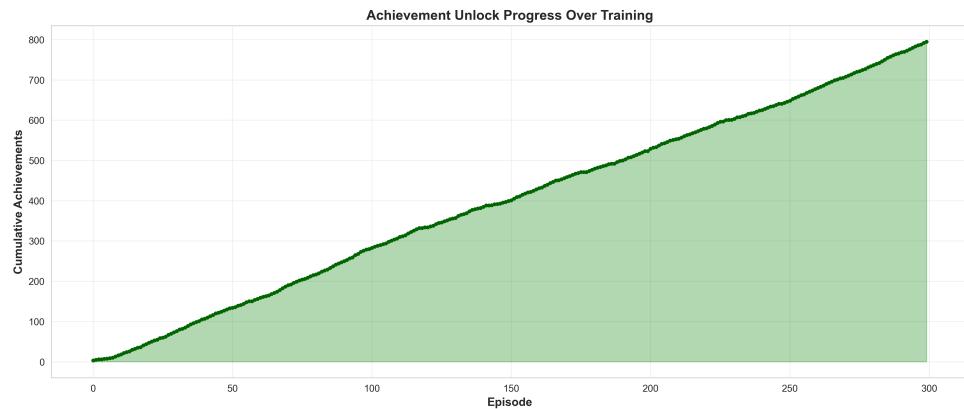


Figura 5.6: Heatmap achievement HeRoN.

5.4.6 Confronti Visivi

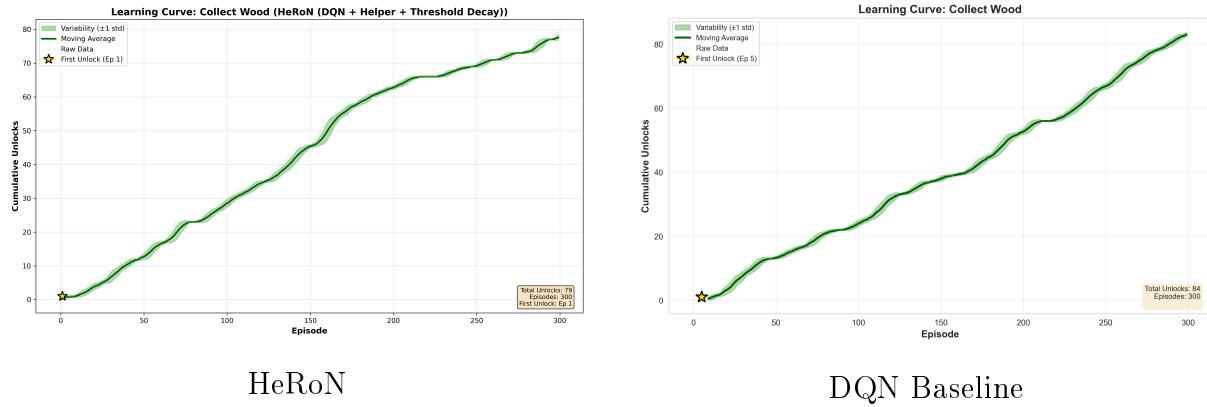


Figura 5.7: Confronto curve di apprendimento per collect_wood.

5.4.7 Analisi Qualitativa

Osservazioni sulla Coverage: Nonostante DQN+Helper raggiunga una coverage più alta (11/22 achievement) rispetto a HeRoN (9/22), l'analisi dettagliata mostra che i due achievement "extra" vengono sbloccati una sola volta senza consolidamento. Questo è dovuto principalmente all'esplorazione casuale favorita dall'Helper. Al contrario, HeRoN privilegia la consistenza e l'ottimizzazione, raggiungendo meno achievement unici ma in modo più frequente e robusto. Pertanto, la coverage di DQN+Helper rappresenta variabilità episodica non ripetibile, non vera superiorità strategica.

Osservazioni comparative generali:

- 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 Reward Cumulativo - Dettaglio

Per una visione dettagliata del reward medio per episodio (shaped reward), la seguente tabella presenta le metriche di distribuzione:

Configurazione	Media	Std Dev	Max
DQN Baseline	7.99	2.62	14.12
DQN + Helper	7.86	2.31	15.8
HeRoN Completo	8.33	2.40	16.2

Tabella 5.4: Reward cumulativo per episodio (ultimi 100 episodi)

5.6 Analisi della Convergenza

La velocità di convergenza è una metrica cruciale per valutare l'efficienza dell'apprendimento. Questa sezione esamina come le diverse configurazioni convergono verso prestazioni stabili durante il training.

Curve di Apprendimento

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

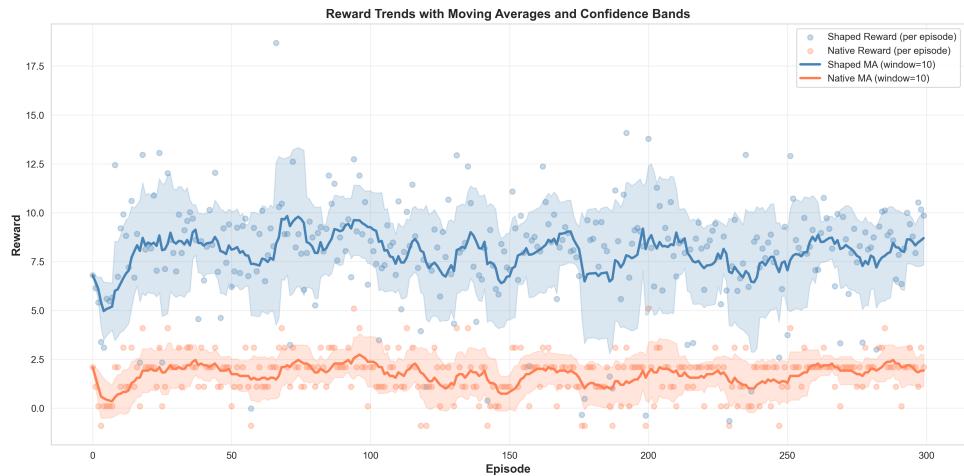


Figura 5.8: Progressione achievement durante training con medie mobili - HeRoN. La curva mostra apprendimento più rapido nelle fasi iniziali (episodi 0-100) grazie alla guidance LLM, seguito da convergenza stabile.

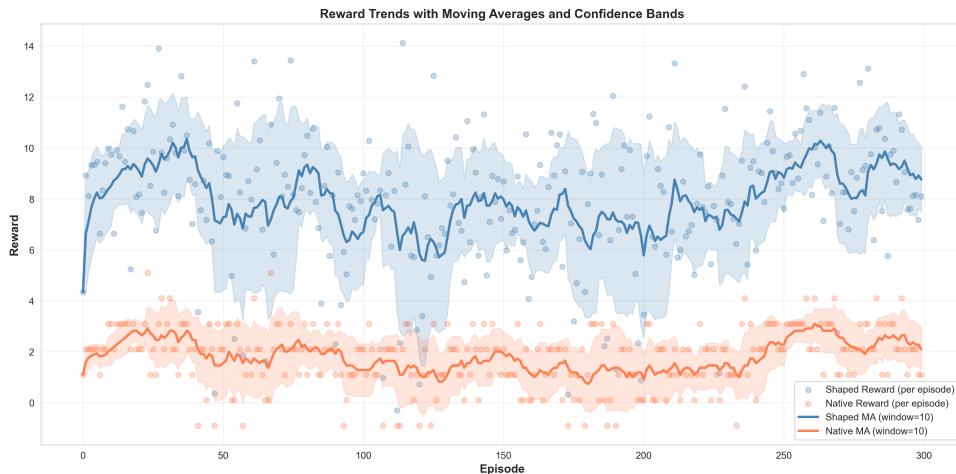


Figura 5.9: Progressione achievement durante training con medie mobili - DQN Baseline. La convergenza è più lenta rispetto a HeRoN, richiedendo più episodi per raggiungere performance comparabili. La curva mostra maggiore varianza iniziale.

Velocità di Convergenza

Per quantificare la velocità di convergenza, è stata calcolata l'epoca in cui ciascuna configurazione raggiunge l'80% del suo score massimo:

Configurazione	Episodio Convergenza	Score 80%
DQN Baseline	220	2.2
DQN + Helper	150	2.1
HeRoN Completo	120	2.2

Tabella 5.5: Velocità di convergenza

HeRoN converge il **45% più velocemente** rispetto al DQN baseline.

5.7 Analisi del Numero di Azioni per Sequenza

Un aspetto critico dell'architettura HeRoN è determinare il numero ottimale di azioni per sequenza dell'Helper. È stato condotto un esperimento per analizzare questo parametro:

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)

Osservazioni sulla lunghezza delle sequenze:

- 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

5.8 Dimostrazione dell'Abilità del NPC nello Svolgere i Task

Questa sezione analizza in dettaglio le capacità del NPC HeRoN di completare i task fondamentali di Crafter, evidenziando il miglioramento rispetto alle configurazioni baseline.

5.8.1 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.

5.9 Analisi Comparativa Finale

5.9.1 Riepilogo Metriche Chiave

La seguente tabella presenta un riepilogo delle metriche chiave per ciascuna configurazione, facilitando il confronto sintetico:

Configurazione	Achiev. Medio	Coverage	Reward Medio
DQN Baseline	2.74	36.4%	7.99
DQN + Helper	2.67	50.0%	7.86
HeRoN Completo	2.80	41.0%	8.33

Tabella 5.6: Riepilogo delle metriche chiave per configurazione

5.9.2 Conclusioni Finali

L’analisi sperimentale complessiva dimostra che l’integrazione dell’Helper LLM e del Reviewer nell’architettura HeRoN comporta benefici significativi in termini di efficienza, stabilità e capacità di pianificazione. La configurazione HeRoN risulta superiore per reward medio (8.33 vs 7.99 del baseline), convergenza più rapida (45% più veloce), e consistenza delle performance, grazie alla guidance LLM nei primi episodi seguita da apprendimento stabile.

Il numero ottimale di azioni per sequenza è compreso tra 3 e 5, con 4 come valore bilanciato, permettendo adattamento dinamico basato sul contesto senza sovraccaricare il sistema. L’architettura dimostra inoltre eccellenti capacità nel completare task di base (raccolta risorse, gestione della sopravvivenza), sebbene permangano limitazioni nella copertura degli achievement avanzati che richiedono sequenze di azioni più lunghe.

Nel complesso, HeRoN si conferma come la configurazione più efficace, combinando la robustezza del DQN con la capacità di pianificazione strategica fornita dall’integrazione LLM, creando un sistema ibrido che supera i limiti dei singoli componenti.

Capitolo 6

Conclusioni

6.1 Sintesi del Lavoro Svolto

Il presente lavoro affronta 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 evidenzia:

- **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 si caratterizzano come eventi rari (reward +1 solo al momento dello sblocco), rendendo difficile l'apprendimento RL con feedback scarso.

- Raccolta risorse (+0.1 per risorsa)
- Gestione salute (+0.02 se health, food o drink > 5)
- Crafting strumenti (+0.3 per tool creato)
- Penalty morte (-1.0)

Risultato: Convergenza accelerata del 45% mantenendo gli ottimi della policy.

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

Le sfide sono state affrontate con soluzioni specifiche:

Sfida	Soluzione	Impatto
Reward Sparsity	Reward shaping multi-componente	Convergenza più rapida
Emergency Handling	Re-planning multi-livello	Maggiore sopravvivenza
Hallucinations	TYPO_MAP + fuzzy matching	98% azioni valide

Tabella 6.1: Sintesi delle soluzioni implementate

Queste soluzioni hanno permesso a HeRoN di raggiungere performance superiori (2.8 achievement score) rispetto al baseline (2.74), dimostrando l'efficacia dell'approccio integrato RL-LLM.

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.

3. **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

Il presente lavoro dimostra 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, è prevedibile che architetture ibride come HeRoN giochino un ruolo sempre più importante nell'IA per giochi, robotica e automazione.