UNIVERSITÀ DEL PIEMONTE ORIENTALE

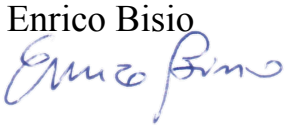
Dipartimento di Scienze e Innovazione Tecnologica

Corso di Laurea Magistrale in Intelligenza Artificiale e Innovazione Digitale

Tesi di Laurea

Modellazione e ottimizzazione di un sistema produttivo mediante Digital Model e Reinforcement Learning

Candidato:
Enrico Bisio



Relatrice:
Prof.ssa Stefania Montani

Anno Accademico 2025/26

Sommario

1. Introduzione

1.1 Contesto industriale e motivazione

Nei sistemi produttivi automatizzati, le prestazioni non dipendono soltanto dalla rapidità delle singole macchine, ma anche dal modo in cui le diverse risorse vengono coordinate. Quando più componenti condividono risorse limitate e operano con tempi diversi, il problema principale consiste nel decidere come assegnare tali risorse e ordinare le operazioni in modo efficiente.

Questa complessità è particolarmente evidente nei sistemi di kitting, in cui oggetti di tipologie differenti devono essere combinati secondo ricette produttive definite. In tali sistemi, le decisioni operative non producono effetti soltanto immediati, ma possono influenzare anche le fasi successive del processo. Una scelta apparentemente conveniente nel breve periodo può infatti generare tempi di attesa, riconfigurazioni o conflitti nell'utilizzo delle risorse; al contrario, una decisione meno vantaggiosa nell'immediato può preparare il sistema a richieste future e ridurre il tempo complessivo di completamento.

Per questo motivo, strategie puramente reattive, basate esclusivamente sulla condizione corrente del sistema, possono risultare subottimali. Per problemi decisionali caratterizzati da azioni sequenziali, vincoli dinamici e conseguenze differite nel tempo, studi recenti sullo scheduling produttivo hanno evidenziato un crescente interesse verso tecniche di Reinforcement Learning e, in particolare, di Deep Reinforcement Learning [1]. Tali tecniche consentono di formulare il problema produttivo come un ambiente decisionale, nel quale un agente apprende politiche di scheduling sulla base della rappresentazione dello stato, dello spazio delle azioni e di un meccanismo di ricompensa.

1.2 Processo produttivo considerato

Il processo produttivo considerato è finalizzato alla realizzazione di kit composti da oggetti di tipologie diverse. Gli oggetti sono inizialmente contenuti in contenitori monotipo, stoccati all'interno di un magazzino verticale. Quando una determinata tipologia è richiesta dal processo, un robot industriale preleva il relativo contenitore dal magazzino e lo trasferisce verso uno degli alimentatori vibranti disponibili.

Gli alimentatori vibranti hanno il compito di rendere disponibili gli oggetti per il riempimento dei kit rilasciandoli singolarmente all'interno dei contenitori posti in corrispondenza del relativo punto di rilascio. Tali contenitori sono trasportati dai mover del sistema XPlanar [2].

Ogni lavorazione segue una ricetta produttiva che definisce quali tipologie di oggetti devono essere inserite e in quali quantità. Nel modello considerato, il sistema serve una ricetta alla volta: una ricetta viene selezionata come attiva e le operazioni successive sono orientate al suo completamento prima di passare alla ricetta seguente. Il completamento di una ricetta richiede quindi il coordinamento di più operazioni: il prelievo dei contenitori monotipo dal magazzino, il loro trasferimento verso gli alimentatori, la disponibilità degli oggetti sui punti di rilascio, il deposito nei contenitori trasportati dai mover e l'avanzamento dei mover lungo il sistema.

Dal punto di vista operativo, la criticità principale riguarda la gestione delle risorse condivise. Poiché gli alimentatori disponibili sono limitati, la scelta della tipologia di oggetto da rendere disponibile deve essere valutata in funzione dello stato complessivo del sistema, e non solo della sua appartenenza alla ricetta. Ogni decisione deve considerare la disponibilità effettiva degli alimentatori, le eventuali riconfigurazioni richieste e l'impatto che tale scelta può avere sull'avanzamento della lavorazione. In questo modo, una decisione locale può influenzare non solo il riempimento di un singolo contenitore, ma anche l'efficienza delle operazioni successive.

1.3 Obiettivo del lavoro

L'obiettivo generale della tesi è verificare se l'applicazione di algoritmi di Reinforcement Learning alla gestione decisionale del processo produttivo considerato possa migliorare le prestazioni produttive del sistema. In particolare, il lavoro valuta se politiche apprese tramite RL siano in grado di ridurre i tempi di completamento delle ricette rispetto a strategie di riferimento basate su regole euristiche.

A tale scopo, è stato sviluppato un modello digitale del processo, utilizzato per addestrare, testare e confrontare le diverse politiche decisionali in condizioni controllate e riproducibili. I risultati ottenuti sono quindi riferiti al modello sviluppato e costituiscono una base per successive verifiche su dati o impianti reali.

2. Fondamenti teorici e modellazione del problema

2.1 Digital Model e simulazione a eventi discreti

La rappresentazione digitale di un sistema produttivo consente di descrivere, analizzare e simulare il comportamento di un processo industriale mediante un modello virtuale. Il grado con cui tale rappresentazione è collegata all'impianto reale può tuttavia variare sensibilmente. Una classificazione proposta in letteratura distingue infatti tre livelli, in funzione dell'integrazione dei dati tra sistema fisico e sistema digitale: Digital Model, Digital Shadow e Digital Twin. Un Digital Model è una rappresentazione digitale di un oggetto o sistema fisico esistente o progettato, priva di scambio automatico di dati con l'impianto fisico. Può quindi essere utilizzato per simulazioni, analisi e valutazione offline, ma non viene aggiornato automaticamente dai dati provenienti dal sistema reale. Un Digital Shadow prevede invece un flusso automatico unidirezionale dal sistema fisico al modello digitale, per cui le variazioni del sistema reale si riflettono nella rappresentazione digitale, ma non viceversa. Un Digital Twin, infine, richiede un'integrazione bidirezionale dei flussi informativi: le variazioni del sistema fisico influenzano il modello digitale e, viceversa, il modello digitale può contribuire ad aggiornare o controllare il comportamento del sistema fisico [3].

Il sistema sviluppato in questo lavoro è quindi classificabile come Digital Model, poiché non è sincronizzato in tempo reale con l'impianto fisico e non acquisisce automaticamente dati dal PLC o dai dispositivi di campo. Il modello riproduce il comportamento logico e temporale del processo produttivo, consentendo di analizzare le decisioni operative, confrontare strategie alternative e generare episodi di simulazione per l'addestramento di politiche di Reinforcement Learning. Tale impostazione è coerente con una fase preliminare di progettazione e sperimentazione, finalizzata a ottenere un ambiente controllabile, verificabile e modificabile, prima di un'eventuale evoluzione verso un Digital Twin operativo.

In particolare, per rappresentare il comportamento temporale del Digital Model, il processo è stato modellato secondo una logica a eventi discreti. Questa scelta è coerente con la natura del sistema analizzato, nel quale l'evoluzione non avviene secondo un

avanzamento temporale continuo o a intervalli uniformi, ma in corrispondenza di eventi che modificano lo stato del processo. Esempi di tali eventi sono il completamento di un'operazione del robot, la disponibilità di un alimentatore vibrante, il rilascio di un oggetto, l'arrivo di un mover in una zona di riempimento o il completamento di una ricetta. La teoria dei Discrete Event Systems fornisce quindi un quadro adatto per descrivere processi produttivi asincroni, nei quali l'evoluzione dello stato è determinata dall'occorrenza di eventi [4]. Questo consente di rappresentare in modo naturale la concorrenza tra risorse, la variabilità dei tempi operativi e l'attivazione delle decisioni solo nei momenti in cui il sistema richiede effettivamente una scelta.

2.2 Formalizzazione del problema decisionale

A partire dal Digital Model introdotto nel paragrafo precedente, il problema affrontato non consiste soltanto nel riprodurre il comportamento del processo produttivo, ma anche nel formalizzare le scelte operative che il sistema deve compiere. Ciò richiede di individuare gli stati in cui è necessario intervenire, le azioni disponibili e il criterio con cui valutarne gli effetti.

Nel sistema considerato, dinamica del processo e decisioni operative rappresentano due aspetti distinti. La prima riguarda il verificarsi di eventi interni, mentre le seconde riguardano le scelte necessarie al coordinamento del sistema. Poiché molte transizioni derivano dalla normale evoluzione temporale del processo, non ogni cambiamento di stato richiede un intervento decisionale. Le decisioni non avvengono quindi a intervalli uniformi, ma in specifici punti decisionali, rendendo il problema riconducibile a un Semi-Markov Decision Process [5].

La complessità del problema deriva dalla presenza di risorse condivise e vincoli dinamici. Alimentatori vibranti, robot e mover interagiscono tra loro: l'impiego di una risorsa può condizionare le scelte successive. Una decisione conveniente nell'immediato può quindi generare attese, riconfigurazioni o conflitti, aumentando il tempo complessivo di esecuzione.

I punti decisionali si articolano su due livelli. A livello globale riguardano la scelta della prossima ricetta da eseguire; a livello locale-operativo riguardano invece la selezione della tipologia di oggetto da assegnare, di volta in volta, al mover. Questa distinzione consente di separare le decisioni di pianificazione da quelle legate all'esecuzione operativa.

Occorre infine definire in modo esplicito lo stato del sistema, le azioni disponibili, il criterio di valutazione e i vincoli che stabiliscono quali azioni siano ammissibili nelle diverse condizioni operative, permettendo così di ricondurre il sistema alla classe dei problemi decisionali sequenziali [6]. Nel modello sviluppato, tali elementi sono costruiti a partire dal Digital Model, che fornisce la base informativa e simulativa necessaria per rappresentare il problema e valutare strategie operative alternative.

2.3 Paradigma di apprendimento adottato

Dato il problema formalizzato nel paragrafo precedente, è necessario adottare un approccio in grado di selezionare azioni in funzione dello stato corrente del sistema, distinguendo scelte ammissibili da quelle non eseguibili, e di valutarne gli effetti sull'evoluzione complessiva del processo. Per questo motivo, la scelta ricade sul Reinforcement Learning, un paradigma di apprendimento in cui un agente interagisce con un ambiente osservandone lo stato, selezionando un'azione e ricevendo una ricompensa, o *reward*, che misura la qualità dell'effetto prodotto. Attraverso la ripetizione di questo ciclo, l'agente aggiorna progressivamente la propria politica decisionale, cioè il criterio con cui associa agli stati osservati le azioni da eseguire, con l'obiettivo di migliorare la prestazione complessiva del sistema [7].

2.3.1 Scelta dell'algoritmo

Gli algoritmi di Reinforcement Learning si distinguono principalmente in metodi *value-based*, che stimano il valore delle azioni, metodi *policy-based*, che apprendono

direttamente una politica di selezione, e metodi *actor-critic*, che combinano una politica con una funzione di valore per guidare l'apprendimento [7].

Gli algoritmi *value-based*, come Q-learning [8] e Deep Q-Network [9], stimano una funzione $Q(s, a)$ che indica quanto sia conveniente scegliere l'azione a nello stato s, considerando anche le ricompense future attese. La politica viene quindi ricavata selezionando l'azione con il valore stimato più alto.

Questi metodi potrebbero essere applicati al caso considerato, ma non nella loro forma standard. Sarebbe infatti necessario modificare il modo in cui l'algoritmo seleziona le azioni e aggiorna la funzione Q: filtrare le azioni solo nella fase di selezione può non essere sufficiente a garantire che l'apprendimento della funzione Q rifletta correttamente i vincoli del problema; approcci *constrained Q-learning* propongono infatti di incorporare tali vincoli direttamente anche nell'aggiornamento della funzione di valore [10]. Per questo motivo, nel contesto considerato, l'approccio *value-based* risulta meno immediato da integrare rispetto a metodi basati direttamente sulla politica.

I metodi *policy-based* apprendono direttamente una politica $\pi(s)$, cioè una distribuzione di probabilità sulle azioni. Un esempio classico è REINFORCE, che non stima il valore di ogni azione, ma aggiorna direttamente la politica aumentando la probabilità delle azioni che hanno prodotto *reward* elevati. Tuttavia, poiché utilizza direttamente i *reward* osservati per aggiornare la politica, può presentare elevata varianza e apprendimento poco stabile, soprattutto in problemi complessi e con segnali di reward ritardati [11]. Tale aspetto risulta rilevante anche nel sistema considerato, in cui gli effetti delle decisioni operative possono manifestarsi solo dopo più eventi successivi.

I metodi *actor-critic* affrontano questo problema affiancando alla politica una stima del valore degli stati. L'*actor* seleziona le azioni secondo la politica appresa, mentre il *critic* valuta lo stato corrente e fornisce un riferimento più stabile per aggiornare la politica, riducendo la dipendenza dai soli *reward* osservati nei singoli episodi [7].

Per questi motivi, nel lavoro svolto è stato adottato un algoritmo della famiglia *actor-critic*. In particolare, la scelta è ricaduta su Proximal Policy Optimization, o PPO, per la sua stabilità nell'aggiornamento della politica e per la possibilità di impiegare una

variante con *action masking*, adatta alla gestione di azioni disponibili variabili in funzione dello stato.

2.3.2 Proximal Policy Optimization e action masking

Proximal Policy Optimization, o PPO, apprende una politica parametrizzata a partire dalle esperienze raccolte durante l'interazione tra agente e ambiente. La sua caratteristica principale è l'introduzione di un aggiornamento controllato della politica, progettato per evitare variazioni troppo ampie tra una versione della politica e la successiva. In questo modo, l'agente può migliorare progressivamente il proprio comportamento senza modificare eccessivamente la strategia appresa ad ogni iterazione [12].

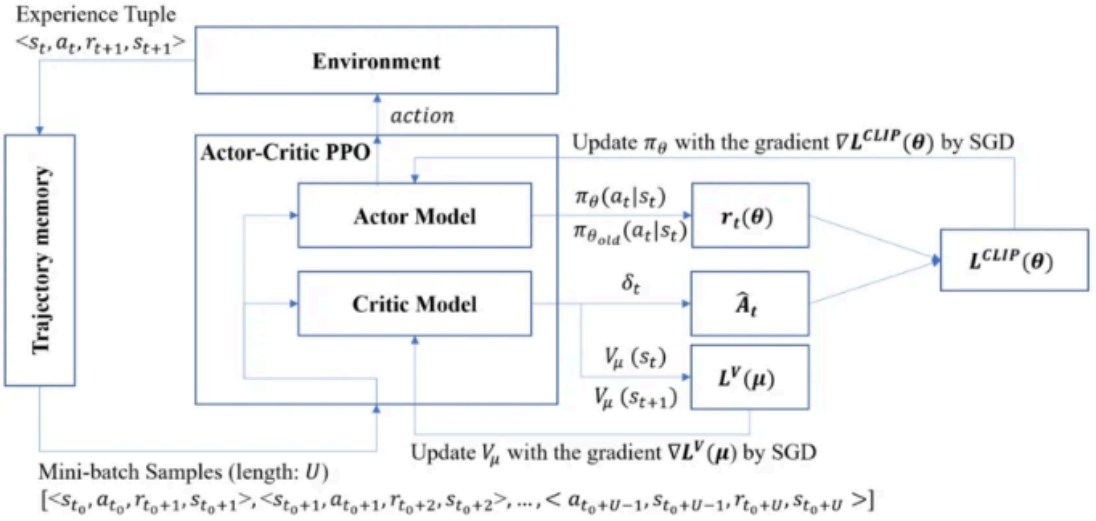


Figura 1 — Processo dell'algoritmo actor–critic PPO. Fonte: Lim et al. [13].

La Figura 1 rappresenta il ciclo di apprendimento di PPO nella formulazione *actor-critic* comunemente adottata. L'agente interagisce con l'ambiente eseguendo

un'azione a_t nello stato corrente s_t. L'ambiente restituisce quindi una transizione, composta dallo stato, dall'azione eseguita, dal *reward* ottenuto e dallo stato successivo, indicata come $\left(s_t, a_t, r_{\{t+1\}}, s_{\{t+1\}}\right)$. Queste transizioni vengono raccolte temporaneamente e successivamente utilizzate per l'addestramento tramite *mini-batch*.

L'*actor* riceve lo stato e produce la politica $\pi_\theta\left(s_t\right)$, cioè la probabilità di selezionare una certa azione nello stato corrente. Durante l'aggiornamento, questa nuova politica viene confrontata con la politica precedente $\pi_{\theta old}\left(s_t\right)$. Dal confronto si ottiene il rapporto $r_t(\theta)$, che misura quanto cambia la probabilità assegnata alla stessa azione nello stesso stato rispetto alla politica precedente.

Il *critic*, invece, stima il valore dello stato, indicato come $V_\mu\left(s_t\right)$. Questa stima viene confrontata con il valore dello stato successivo $V_\mu\left(s_{t+1}\right)$ e con il *reward* osservato, producendo un errore temporale δ_t. Da questo errore viene calcolato il vantaggio stimato $\hat{A}_t$, che indica se l'azione scelta ha prodotto un risultato migliore o peggiore rispetto a quanto atteso.

Il vantaggio $\hat{A}_t$ e il rapporto $r_t(\theta)$ vengono successivamente usati nella funzione obiettivo clipped $L^{CLIP}(\theta)$. Questa funzione limita l'aggiornamento della politica quando la nuova distribuzione si discosta eccessivamente da quella precedente. In parallelo, il *critic* viene aggiornato tramite la propria funzione di perdita $L^V(\mu)$, con l'obiettivo di migliorare la stima del valore degli stati. PPO coordina quindi l'aggiornamento della politica e della funzione di valore mantenendo controllata l'evoluzione della strategia appresa [12].

Nel problema considerato, l'applicazione di PPO è completata dall'uso dell'*action masking*. Questa tecnica applica una maschera alla distribuzione di probabilità prodotta dalla politica, escludendo le azioni non valide prima della selezione dell'azione. L'agente non campiona quindi dall'intero spazio delle azioni, ma solo dal sottoinsieme disponibile nello stato corrente. L'*invalid action masking* è particolarmente utile nei

problemi con spazi d'azione discreti e vincoli dipendenti dallo stato, perché impedisce la selezione delle azioni non valide prima dell'esecuzione, invece di lasciarle scegliere all'agente e penalizzarle solo successivamente tramite il *reward* [14].

3. Sviluppo del Digital Model e integrazione del Reinforcement Learning

3.1 Architettura generale del software

Come mostrato in Figura 2, il software sviluppato è stato organizzato secondo una struttura modulare articolata in tre parti principali: modello del processo produttivo, dispatcher e modulo decisionale. Questa suddivisione consente di separare la simulazione del sistema, il coordinamento delle risorse e la scelta delle azioni da applicare, evidenziando i flussi informativi tra modello simulativo, dispatcher e modulo decisionale.

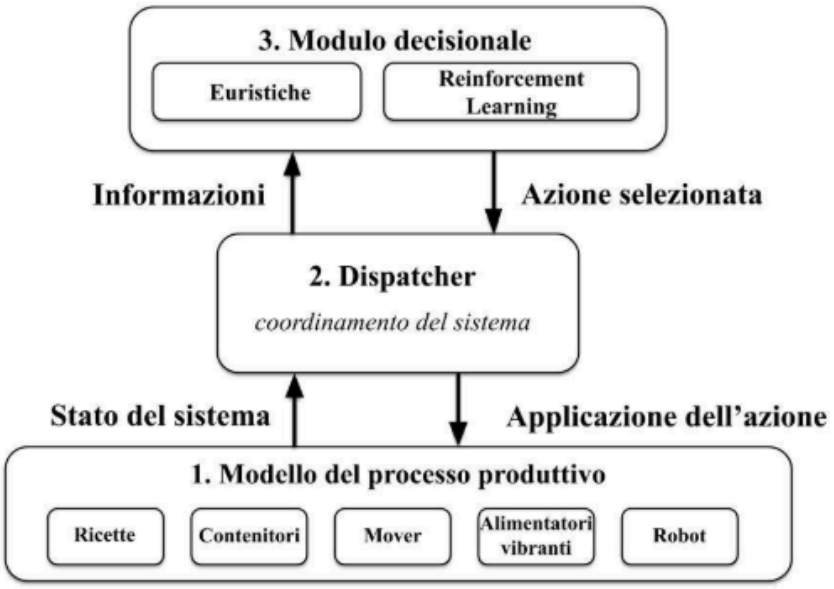


Figura 2 — Architettura generale del software

Il modello del processo produttivo costituisce la base dell'architettura. Esso comprende le classi che rappresentano le principali componenti simulate del sistema, tra cui contenitori, mover, alimentatori vibranti e robot, insieme alle strutture dati che descrivono le ricette produttive. Le componenti simulate definiscono lo stato operativo

del sistema e ne determinano l'evoluzione durante la simulazione, mentre le ricette specificano le tipologie e le quantità di oggetti richieste per il completamento dei contenitori.

Il dispatcher rappresenta il livello di coordinamento. Il suo compito è osservare lo stato del modello, assegnare i task ai mover disponibili, verificare la disponibilità degli alimentatori e gestire le operazioni necessarie all'avanzamento del processo, come il caricamento, lo svuotamento o la riconfigurazione delle risorse. Quando il sistema raggiunge una condizione che richiede una scelta, il dispatcher apre un punto decisionale e fornisce al modulo decisionale le informazioni sullo stato corrente e sulle azioni ammissibili. L'azione selezionata viene quindi restituita al dispatcher, che la applica al modello e permette alla simulazione di proseguire.

Il modulo decisionale seleziona le azioni da applicare nei momenti in cui il dispatcher richiede una scelta. Esso utilizza le informazioni operative disponibili e l'insieme delle azioni ammissibili per selezionare l'azione più opportuna. Tale scelta può essere determinata mediante strategie differenti, come regole euristiche oppure politiche apprese tramite Reinforcement Learning. La separazione tra simulazione e decisione consente quindi di confrontare diverse politiche decisionali mantenendo invariata la logica simulativa di base.

3.2 Architettura decisionale gerarchica

Nel processo produttivo considerato è possibile distinguere due livelli decisionali complementari. Il primo livello, di natura globale, riguarda la selezione della ricetta da eseguire tra quelle disponibili; il secondo, di tipo operativo, riguarda invece le decisioni necessarie durante l'esecuzione della ricetta attiva, in particolare la scelta della tipologia di oggetto da assegnare al mover coinvolto nel punto decisionale.

Sulla base di questa distinzione, il sistema decisionale è stato strutturato secondo un'architettura gerarchica a due livelli. Il livello globale opera su una scala temporale più ampia e determina la sequenza delle ricette da processare. La decisione globale

definisce quindi il contesto operativo entro cui agisce il livello operativo. Quest'ultimo interviene invece su una scala temporale più fine, nei decision boundary locali individuati dal dispatcher, scegliendo quale tipologia di oggetto servire in funzione dello stato corrente del sistema.

Questa impostazione è coerente con il principio dello Hierarchical Reinforcement Learning, nel quale un problema decisionale complesso viene scomposto in sottoproblemi più semplici, organizzati secondo diversi livelli di astrazione [15].

Nel modello sviluppato, l'interazione tra i due livelli avviene in modo sequenziale. Quando il sistema richiede una nuova ricetta, il livello globale seleziona l'alternativa da attivare tra quelle ancora non completate. Una volta definita la ricetta attiva, il controllo passa al livello operativo, che prende una serie di decisioni locali fino al completamento della ricetta stessa. Al termine dell'esecuzione, il sistema ritorna al livello globale per selezionare la ricetta successiva.

Questa suddivisione giustifica l'impiego di due politiche di Reinforcement Learning distinte. I due livelli operano infatti su spazi di osservazione, spazi di azione e scale temporali differenti: il livello globale osserva lo stato complessivo delle ricette e sceglie quale ricetta attivare, mentre il livello operativo osserva lo stato della ricetta attiva, dei mover e delle risorse disponibili per scegliere quale tipologia di oggetto servire. Per questo motivo sono state addestrate due reti separate: una politica globale, dedicata alla selezione delle ricette, e una politica operativa, dedicata alle decisioni locali durante l'esecuzione della ricetta attiva.

La separazione in due reti consente di specializzare ciascuna politica sul proprio sottoproblema decisionale, evitando che un'unica rete debba apprendere simultaneamente decisioni eterogenee per significato, frequenza e orizzonte temporale. La decomposizione gerarchica rende quindi il problema più strutturato: il livello globale si concentra sulle decisioni strategiche di sequenziamento, mentre il livello operativo ottimizza l'esecuzione locale della ricetta corrente.

3.3 Modellazione a eventi discreti con SimPy

La scelta del framework è stata effettuata valutando diverse alternative. Strumenti industriali come AnyLogic e Arena offrono funzionalità avanzate: AnyLogic include esperimenti per l'integrazione con agenti di Reinforcement Learning [16], mentre Arena è orientato alla simulazione a eventi discreti di processi industriali [17]. Tuttavia, è stata preferita una soluzione open-source e integrata in Python, basata sul framework SimPy [18], per mantenere maggiore controllo sul modello e facilitare il collegamento con il modulo decisionale.

3.3.1 Caratteristiche di SimPy

SimPy è un framework open-source per la simulazione a eventi discreti sviluppato in Python. Esso consente di rappresentare il comportamento di un sistema mediante processi, definiti tramite funzioni generatrici Python, che possono sospendere e riprendere la propria esecuzione in corrispondenza di eventi [19]. Inoltre, mette a disposizione strumenti di base per la gestione di eventi, risorse condivise a capacità limitata, cioè risorse che possono essere occupate solo da un numero limitato di processi alla volta, e strutture di coda utilizzabili per modellare l'interazione tra i processi durante la simulazione [20]. Nel modello sviluppato, tali strumenti consentono di rappresentare l'accesso controllato alle risorse fisiche, le code operative di comandi e task e i punti in cui la simulazione deve sospendersi in attesa di una decisione esterna.

Il funzionamento di SimPy si basa su un ambiente che gestisce l'orologio della simulazione e organizza gli eventi in una coda ordinata. Ogni evento è associato a un tempo, a una priorità e a un identificativo progressivo; tali informazioni stabiliscono l'ordine con cui gli eventi vengono processati, anche nel caso di eventi programmati nello stesso istante [22].

Quando un processo deve attendere il verificarsi di un evento, la sua esecuzione viene sospesa tramite l'istruzione yield. Il processo rimane quindi in attesa finché l'evento non viene elaborato dall'ambiente di simulazione. Al verificarsi dell'evento, SimPy

aggiorna il tempo simulato, riattiva i processi sospesi e, se previsto, trasmette loro il valore prodotto dall'evento [21].

La simulazione non procede secondo un passo temporale fisso, ma avanza direttamente da un evento rilevante al successivo. Questo consente di evitare l'elaborazione di intervalli temporali in cui lo stato del sistema non cambia e rende più efficiente la modellazione di processi asincroni.

Un ulteriore aspetto rilevante è la possibilità di controllare esplicitamente l'avanzamento del modello. SimPy consente di eseguire la simulazione fino all'esaurimento degli eventi, fino a un determinato istante, fino al verificarsi di uno specifico evento oppure avanzando manualmente evento per evento [22]. Nel lavoro svolto, questa caratteristica permette di far evolvere il Digital Model fino a un punto decisionale, sospendere temporaneamente l'esecuzione, applicare l'azione scelta dal modulo decisionale e proseguire fino alla decisione successiva.

3.3.2 Implementazione del Digital Model in SimPy

La costruzione del Digital Model avviene nel modulo `factory.py`, attraverso la funzione `make_dispatcher`. Essa costituisce il punto di inizializzazione del modello simulativo: crea l'ambiente di simulazione SimPy, carica le ricette da file `JSON`, determina le tipologie di oggetti presenti e istanzia le principali componenti del sistema, tra cui alimentatori vibranti, mover, robot e dispatcher. La funzione restituisce quindi l'ambiente SimPy e il dispatcher, lasciando al codice chiamante il controllo dell'esecuzione. In questo modo, inizializzazione e avanzamento della simulazione rimangono separati, consentendo di utilizzare lo stesso modello in contesti differenti, ad esempio per eseguire simulazioni complete, verificare il comportamento delle componenti o arrestare l'esecuzione al raggiungimento di specifiche condizioni operative.

Tutte le componenti condividono lo stesso `simpy.Environment,` che rappresenta il tempo simulato comune e coordina l'esecuzione dei processi. Gli alimentatori vibranti,

descritti dalla classe `Vibrator`, rappresentano le risorse incaricate del rilascio degli oggetti verso i contenitori trasportati dai mover. Ciascun alimentatore mantiene il proprio stato operativo, comprendente la tipologia caricata, la quantità residua, lo stato di occupazione, eventuali condizioni di fault e la presenza di un mover in posizione di riempimento. Queste informazioni consentono al dispatcher di valutare se l'alimentatore possa servire una determinata tipologia oppure se sia necessario un intervento del robot, ad esempio per il caricamento o la riconfigurazione della risorsa.

I mover, implementati nella classe `Mover`, rappresentano le unità di trasporto che movimentano i contenitori in lavorazione. Ciascun mover può essere associato a un contenitore e ricevere un task di riempimento, cioè l'insieme delle quantità richieste per ciascuna tipologia di oggetto in relazione a una specifica ricetta. Durante la simulazione, il mover controlla il fabbisogno residuo del contenitore associato e interagisce con gli alimentatori per ricevere gli oggetti necessari al completamento del task.

Il robot, definito nella classe `Robot`, rappresenta la risorsa incaricata delle operazioni di caricamento, svuotamento e riconfigurazione degli alimentatori vibranti. Nel modello tali operazioni sono gestite tramite una coda SimPy di comandi ed eseguite in sequenza, introducendo tempi di servizio e costi di riconfigurazione dipendenti dalla transizione tra tipologie di oggetti.

Il modello introduce, inoltre, dinamiche temporali dipendenti dallo stato del sistema. I tempi di servizio variano in funzione della tipologia di oggetto gestita e il cambio di tipologia su un alimentatore può richiedere un'operazione di riconfigurazione. Di conseguenza, l'effetto di una decisione non dipende soltanto dall'azione scelta, ma anche dalla configurazione corrente delle risorse, dalla disponibilità del robot e dal fabbisogno residuo dei contenitori.

Questa impostazione rende la simulazione più realistica rispetto a una semplice sequenza di operazioni indipendenti: una scelta apparentemente conveniente nell'immediato può generare attese o riconfigurazioni successive, mentre una scelta meno diretta può risultare più vantaggiosa sul tempo complessivo di completamento.

Il coordinamento generale è affidato al dispatcher, che gestisce lo stato operativo del sistema, assegna i task ai mover disponibili, controlla l'avanzamento della ricetta attiva e verifica la disponibilità degli alimentatori. A livello operativo, quando un mover richiede una nuova tipologia di oggetto, il dispatcher determina l'insieme delle azioni ammissibili sulla base dello stato corrente del sistema. In questa fase vengono considerate le richieste residue del contenitore, la ricetta attiva e la disponibilità degli alimentatori.

Il punto decisionale viene gestito tramite un evento SimPy, che consente di sospendere temporaneamente l'esecuzione e di rendere disponibili al modulo decisionale le informazioni necessarie alla scelta. Una volta selezionata l'azione, il dispatcher la applica al modello simulativo, attivando se necessario comandi verso il robot o aggiornamenti dello stato degli alimentatori. Dopo l'applicazione della decisione, la simulazione riprende secondo la normale dinamica a eventi discreti fino al successivo punto decisionale.

3.4 Integrazione con il Reinforcement Learning

Per integrare il Reinforcement Learning con il Digital Model sviluppato è stata utilizzata la libreria Stable-Baselines3, che fornisce implementazioni affidabili di algoritmi di Reinforcement Learning basati su PyTorch, insieme a strumenti per l'addestramento, la valutazione, il logging, il salvataggio dei modelli e l'utilizzo di ambienti personalizzati [23]. Questa scelta consente di utilizzare algoritmi già validati e ampiamente adottati, concentrando il lavoro sull'integrazione con il modello simulativo, sulla definizione delle osservazioni, delle azioni e della funzione di reward.

Poiché il problema presenta azioni discrete ma non sempre ammissibili in ogni stato del sistema, è stata adottata la variante `MaskablePPO`, disponibile nel pacchetto SB3-Contrib. Tale variante estende PPO introducendo il supporto all'*action masking*, cioè un meccanismo che consente di limitare la selezione dell'agente alle sole azioni valide nello stato corrente [24].

3.4.1 Caratteristiche di Gymnasium

Per poter utilizzare Stable-Baselines3, il simulatore deve essere reso accessibile all'algoritmo attraverso un'interfaccia standard. A questo scopo è stata adottata Gymnasium, una libreria Python open-source che definisce una API comune per la costruzione di ambienti di Reinforcement Learning [25]. Gymnasium non rappresenta un simulatore autonomo, ma fornisce la struttura attraverso cui un algoritmo di apprendimento può interagire con un ambiente, osservandone lo stato, selezionando azioni e ricevendo una misura dell'effetto prodotto.

Il funzionamento di un ambiente Gymnasium si basa principalmente sui metodi `reset` e `step`. Il metodo `reset` inizializza un nuovo episodio e restituisce la prima osservazione dell'ambiente, mentre `step` riceve un'azione, aggiorna l'ambiente e restituisce le informazioni necessarie al ciclo di apprendimento: la nuova osservazione, la reward, gli indicatori di terminazione o troncamento dell'episodio e un dizionario `info` utilizzato per fornire informazioni aggiuntive di monitoraggio che non fanno parte dell'osservazione dell'agente [26].

Gymnasium richiede inoltre di definire lo spazio delle osservazioni e lo spazio delle azioni. Lo spazio delle osservazioni specifica il formato dello stato osservabile restituito dall'ambiente, ad esempio un vettore numerico, una matrice o una struttura più complessa. Lo spazio delle azioni descrive invece l'insieme delle scelte disponibili per l'agente, che possono essere discrete, continue o composte [27].

3.4.2 Implementazione degli ambienti Gymnasium

L'integrazione tra Digital Model e Reinforcement Learning è realizzata mediante due ambienti Gymnasium distinti, coerenti con l'architettura decisionale gerarchica: `RuntimeDispatchEnv` e `GlobalEnv`. Il primo è dedicato al livello operativo, nel quale l'azione corrisponde alla scelta della tipologia di oggetto da assegnare al mover coinvolto nel punto decisionale. Il secondo è dedicato al livello globale, nel quale l'azione corrisponde alla selezione della prossima ricetta da eseguire.

I due ambienti condividono la stessa logica generale di interazione con la simulazione, ma differiscono nel modo in cui viene inizializzato il modello. In `RuntimeDispatchEnv`, l'agente non deve scegliere la ricetta da eseguire: la ricetta rappresenta il contesto dell'episodio, mentre la decisione riguarda la tipologia di oggetto da assegnare al mover. Per questo motivo, la simulazione viene costruita tramite una funzione esterna, `make_dispatcher_fn`, fornita dal codice di esecuzione e richiamata nel metodo `reset`. In questo modo il codice di training può stabilire quale scenario o ricetta utilizzare, mentre l'ambiente operativo rimane concentrato sulla scelta della tipologia per il mover.

In `GlobalEnv`, invece, la decisione dell'agente riguarda il livello globale, cioè la scelta della ricetta da eseguire tra quelle disponibili nello scenario simulato. L'ambiente riceve un insieme di file di ricette e, a ogni `reset`, seleziona uno scenario da utilizzare per costruire la simulazione tramite `make_dispatcher`. Le ricette caricate nel dispatcher costituiscono quindi l'insieme delle alternative tra cui l'agente globale può scegliere.

Anche il metodo `step` segue una struttura comune nei due ambienti. L'azione ricevuta dall'agente è un indice numerico: in `RuntimeDispatchEnv` identifica una tipologia di oggetto, mentre in `GlobalEnv` identifica una ricetta selezionabile. Una volta selezionata l'azione, l'ambiente completa l'evento decisionale aperto nella simulazione e invia la decisione al dispatcher, che la applica al modello simulativo. La simulazione viene quindi fatta avanzare tramite SimPy fino al successivo punto decisionale oppure fino alla conclusione dell'episodio. La sequenza complessiva è sintetizzata in Figura 3, che mostra il ciclo di interazione tra ambiente Gymnasium, dispatcher e simulazione SimPy, dall'inizializzazione del modello alla selezione dell'azione, fino alla restituzione degli output previsti dall'interfaccia dell'ambiente.

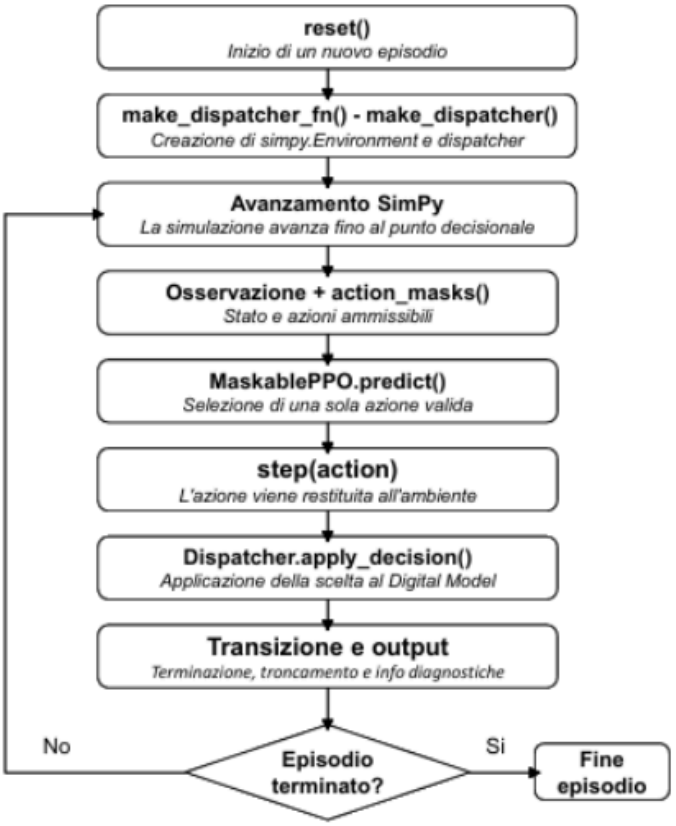


Figura 3 — Ciclo di interazione tra ambiente Gymnasium, dispatcher e simulazione SimPy.

In entrambi gli ambienti, lo stato interno del modello viene trasformato in un'osservazione numerica a dimensione fissa, compatibile con lo spazio di osservazione Gymnasium. Nel livello operativo, l'osservazione è centrata sul mover coinvolto nella decisione e sullo stato della ricetta attiva; nel livello globale, invece, descrive lo stato complessivo delle ricette e delle risorse.

Gli ambienti implementano inoltre il metodo `action_masks`, che restituisce una maschera booleana delle azioni ammissibili nello stato corrente. In `RuntimeDispatchEnv`, la maschera specifica quali tipologie di oggetto possono essere selezionate nel punto decisionale operativo corrente. In `GlobalEnv`, invece, identifica le ricette ancora selezionabili; in questo caso lo spazio delle azioni include anche una posizione aggiuntiva di fallback, utilizzata per evitare che la maschera risulti

priva di azioni valide. Tale posizione non rappresenta una ricetta produttiva effettiva, ma una scelta di sicurezza introdotta per mantenere valida l'interfaccia dell'ambiente anche nei casi limite. La maschera viene quindi fornita a `MaskablePPO`, che limita la selezione dell'agente alle sole azioni compatibili con lo stato corrente del sistema.

La reward viene calcolata dagli ambienti al termine di ogni transizione decisionale, osservando l'evoluzione del modello tra due punti decisionali consecutivi. Essa rappresenta il segnale numerico restituito all'algoritmo per valutare l'effetto dell'azione selezionata.

Gli ambienti restituiscono inoltre informazioni diagnostiche tramite il dizionario info, utili per monitorare l'avanzamento della simulazione, le condizioni di terminazione e i principali indicatori dell'episodio. Tali informazioni non fanno parte dell'osservazione dell'agente, ma supportano le attività di debug, validazione e valutazione sperimentale.

3.4.3 Costruzione delle osservazioni locale e globale

La costruzione dell'osservazione rappresenta un passaggio centrale nell'integrazione tra Digital Model e Reinforcement Learning, poiché stabilisce quali informazioni dello stato simulato vengono rese disponibili all'agente nel momento in cui deve prendere una decisione. L'osservazione non contiene quindi l'intero stato interno della simulazione, ma solo le informazioni utili a scegliere l'azione più adatta nel punto decisionale corrente.

Nel livello locale, implementato tramite `RuntimeDispatchEnv`, l'osservazione combina informazioni locali e di contesto. Le informazioni locali servono a descrivere il bisogno immediato del mover coinvolto nella decisione, mentre quelle di contesto permettono di valutare se la scelta è coerente con lo stato della ricetta attiva e con la disponibilità delle risorse. In questo modo l'agente non sceglie la tipologia da servire solo in base al contenitore corrente, ma anche in base alle condizioni operative del sistema.
La prima informazione fornita all'agente è il fabbisogno residuo del contenitore

associato al mover. Per ogni tipologia di oggetto viene indicata la quantità ancora richiesta dal contenitore. Questo dato serve a identificare quali tipologie sono effettivamente utili per far avanzare il mover corrente. Il valore è normalizzato rispetto alla quantità prevista per quella stessa tipologia nel task del contenitore, così da rendere confrontabili richieste di dimensione diversa. Se una tipologia non è richiesta dal contenitore, il valore corrispondente viene posto a zero, indicando che quella scelta non contribuirebbe direttamente al completamento del contenitore.

Alle informazioni locali si aggiunge il fabbisogno residuo globale della ricetta attiva. Questo valore indica quanto ciascuna tipologia sia ancora rilevante per completare l'intera ricetta. Serve quindi a dare all'agente una visione più ampia rispetto al singolo mover: una tipologia può essere utile al contenitore corrente, ma può essere anche più o meno importante per l'avanzamento complessivo della ricetta. Il valore è normalizzato rispetto alla quantità inizialmente richiesta dalla ricetta per quella stessa tipologia, in modo da esprimere l'importanza relativa della tipologia nella lavorazione ancora da completare.

L'osservazione locale include inoltre lo stato delle risorse disponibili, in particolare degli alimentatori vibranti. Questa informazione serve a distinguere le tipologie che possono essere servite immediatamente da quelle che richiederebbero caricamenti, attese o riconfigurazioni. La disponibilità è codificata tramite variabili binarie: il valore 1 indica che una condizione è verificata, mentre il valore 0 indica che non lo è. In questo modo l'agente può valutare non solo cosa sarebbe utile servire, ma anche cosa è conveniente servire nello stato corrente del sistema.

Infine, viene inclusa un'informazione temporale relativa al tempo simulato trascorso tra due punti decisionali consecutivi. Questo dato serve a far percepire all'agente il costo temporale delle transizioni: due decisioni possono produrre un avanzamento simile, ma richiedere tempi diversi. Nel codice, il tempo viene diviso per 10.0 e poi trasformato tramite la funzione $tanh$; poiché il tempo è non negativo, il valore risultante rimane compreso tra 0 e 1. In questo modo la rete può tenere conto della durata delle scelte senza che il tempo domini le altre componenti dell'osservazione.

Nel livello globale, implementato tramite `GlobalEnv`, l'osservazione descrive lo stato complessivo dello scenario produttivo. A differenza del livello operativo, non è riferita a un singolo mover, ma alla scelta della prossima ricetta da eseguire. Per questo motivo, il vettore di osservazione fornisce all'agente informazioni sulle ricette ancora presenti nello scenario e sulla configurazione corrente degli alimentatori vibranti.

La prima parte dell'osservazione globale rappresenta la configurazione degli alimentatori vibranti. Per ciascun alimentatore viene indicata la tipologia di oggetto attualmente caricata mediante una codifica binaria per tipologia: il valore 1 indica che l'alimentatore contiene una determinata tipologia, mentre il valore 0 indica che quella tipologia non è presente. Questa informazione serve all'agente globale per sapere quali tipologie sono già disponibili nel sistema prima di scegliere la prossima ricetta.

La seconda parte dell'osservazione descrive lo stato delle ricette. Per ogni ricetta viene inserito un valore binario che indica se essa è ancora da completare oppure è già terminata: il valore 1 indica una ricetta non ancora completata, mentre il valore 0 indica una ricetta conclusa. Questo dato serve a distinguere le ricette ancora utili da quelle che non devono più essere selezionate.

Per ciascuna ricetta ancora da eseguire viene poi rappresentato il fabbisogno residuo per tipologia, normalizzato rispetto al totale residuo della ricetta. Questa informazione descrive la composizione della ricetta, cioè quali tipologie richiede e con quale peso relativo. Serve quindi all'agente per confrontare le ricette non solo come alternative disponibili, ma anche in base al tipo di lavoro che richiedono.

A partire da questa composizione viene calcolata anche una misura di compatibilità con la configurazione corrente degli alimentatori. Questo valore indica quanta parte del fabbisogno della ricetta riguarda tipologie già presenti sugli alimentatori. Serve quindi a valutare quanto una ricetta sia adatta allo stato attuale delle risorse: una compatibilità più alta indica che la ricetta può sfruttare meglio gli alimentatori già configurati e può richiedere meno riconfigurazioni o tempi di attesa. In questo modo l'agente globale può scegliere la prossima ricetta considerando non solo quali ricette restano da completare, ma anche quale risulta più conveniente rispetto alla configurazione corrente del sistema.

4. Addestramento e valutazione sperimentale

4.1 Verifica e validazione del Digital Model

La verifica e la validazione del Digital Model costituiscono una fase preliminare necessaria prima del suo utilizzo come ambiente di addestramento e valutazione delle politiche decisionali. L'impostazione adottata in questo lavoro segue il metodo proposto da Sargent per la verifica e validazione dei modelli di simulazione: la verifica riguarda il controllo della corretta traduzione del modello concettuale nel programma di simulazione, mentre la validazione riguarda il controllo che il comportamento della simulazione sia coerente con quello atteso per il sistema rappresentato [28].

Nel lavoro svolto, la verifica dell'implementazione è stata condotta mediante test automatici e simulazioni controllate. Sono stati controllati il caricamento delle ricette da file JSON, l'inizializzazione dell'ambiente SimPy, la creazione delle risorse simulate, l'assegnazione dei task ai mover e il comportamento delle principali componenti del modello: dispatcher, robot, alimentatori vibranti e mover. Sono stati inoltre verificati l'avanzamento del tempo simulato, la gestione degli eventi e l'assenza di anomalie strutturali della simulazione, quali blocchi, deadlock, livelock o cicli senza progresso effettivo.

Per rendere più robusta l'esecuzione automatica delle simulazioni, nell'ambiente sono stati introdotti alcuni limiti di sicurezza, relativi al numero massimo di eventi interni elaborabili tra due decisioni consecutive e al numero massimo di eventi consecutivi senza avanzamento del tempo simulato. Tali limiti non descrivono proprietà del processo produttivo, ma controlli software introdotti nel codice per intercettare situazioni anomale, come blocchi, cicli senza progresso o esecuzioni non produttive, che durante l'addestramento potrebbero impedire la corretta conclusione degli episodi.

È stata inoltre verificata la corretta gestione dei punti decisionali. In particolare, è stato controllato che essi venissero generati solo quando il sistema richiedeva effettivamente una scelta, che l'osservazione fornita all'agente fosse coerente con lo stato interno del modello, che la maschera delle azioni includesse solo azioni ammissibili e che, dopo

l'applicazione dell'azione selezionata, la simulazione proseguisse fino al successivo punto decisionale.

La validazione logica e comportamentale è stata condotta mediante simulazioni complete del processo. È stato verificato che il modello riproducesse la sequenza operativa attesa: selezione della ricetta attiva, assegnazione dei contenitori ai mover, richiesta delle tipologie necessarie, eventuale caricamento o riconfigurazione degli alimentatori, rilascio degli oggetti e completamento della ricetta. È stata inoltre controllata la coerenza del comportamento delle risorse condivise, verificando che robot, alimentatori e mover producessero effetti plausibili sul processo, come attese, riconfigurazioni e variazioni del tempo di completamento in funzione della sequenza delle decisioni.

Poiché il modello sviluppato è un Digital Model e non è collegato all'impianto fisico, non è stata effettuata una validazione quantitativa tramite confronto con dati reali. La validazione è stata quindi condotta rispetto allo scopo dello studio, in accordo con Sargent, secondo cui un modello di simulazione deve essere valutato in funzione dell'applicazione per cui viene costruito e utilizzato [28]. Nel caso specifico, l'obiettivo è disporre di un ambiente controllato e riproducibile per addestrare, testare e confrontare politiche decisionali. Pertanto, il modello può essere considerato adeguato allo scopo della tesi, mentre i risultati ottenuti vanno considerati riferiti al modello simulato e non direttamente estendibili all'impianto reale senza ulteriori attività di calibrazione e validazione sperimentale.

4.2 KPI di valutazione

La valutazione delle politiche decisionali si è basata su due indicatori principali: il completamento della ricetta e il tempo di completamento.

Nel sistema reale la ricetta è progettata per arrivare a completamento. Nella simulazione, invece, è necessario verificare esplicitamente che l'episodio si concluda correttamente, poiché l'ambiente può arrestarlo prima della fine della lavorazione

quando vengono raggiunti i limiti massimi previsti, ad esempio durante esecuzioni non produttive o troppo lunghe. In questi casi il tempo registrato non rappresenta un vero tempo di completamento e non può essere confrontato con quello di una politica che conclude correttamente la ricetta. Per questo motivo, il completamento è utilizzato come criterio preliminare di validità della politica.

Il secondo KPI è il tempo di completamento, cioè il tempo simulato necessario per concludere correttamente la lavorazione. Questo indicatore rappresenta la metrica principale di ottimizzazione: tra le politiche che completano la ricetta, viene considerata migliore quella che ottiene il tempo minore. Il tempo simulato consente quindi di valutare l'efficacia delle decisioni nel coordinamento delle risorse condivise.

Il criterio di confronto adottato è quindi gerarchico. In primo luogo viene verificato il completamento corretto della ricetta o, nel caso del livello globale, dell'insieme di ricette previste dallo scenario. Solo successivamente, tra le politiche che soddisfano questa condizione, viene confrontato il tempo simulato di esecuzione. Questa impostazione evita di confrontare episodi conclusi correttamente con episodi arrestati prima del completamento.

Gli stessi KPI sono stati applicati in modo coerente durante l'intera valutazione sperimentale. Questo ha permesso di confrontare le prestazioni delle singole politiche e, successivamente, di analizzare l'effetto della loro integrazione nell'architettura gerarchica complessiva.

4.3 Funzione di reward

Sulla base dei KPI definiti, la funzione di reward è stata progettata per tradurre gli obiettivi di completamento e riduzione del tempo di esecuzione in un segnale numerico utilizzabile durante l'addestramento degli agenti.

La progettazione è stata articolata sui due livelli decisionali del sistema. In entrambi i casi, la reward viene calcolata a ogni transizione decisionale, confrontando lo stato del sistema prima e dopo l'azione selezionata. In questo modo, l'agente riceve

un'informazione legata sia all'avanzamento della lavorazione sia al tempo simulato trascorso.

La scelta dei pesi è stata effettuata mediante prove empiriche, supportate da script di ottimizzazione basati su Optuna [29], una libreria Python per l'ottimizzazione automatica degli iperparametri.

4.3.1 Reward operativa

La reward a livello operativo è progettata per guidare l'agente verso il completamento rapido della ricetta attiva. Essa combina un termine positivo legato al progresso della lavorazione con penalità associate al tempo trascorso e all'assenza di avanzamento utile.

La funzione di reward è definita come:

$$r_k = w_P \cdot \Delta P_k - w_t \cdot \Delta t_k - w_s \cdot S_k + w_{term} \cdot T_k$$

dove:

- ΔP_k rappresenta la riduzione del lavoro residuo della ricetta tra due decisioni consecutive e costituisce il principale contributo positivo della reward. Il valore viene normalizzato rispetto al lavoro iniziale della ricetta, in modo che il progresso sia espresso in forma relativa e non dipenda direttamente dalla quantità totale di oggetti richiesti. È associato a un peso pari a 5.0, che assegna maggiore importanza alle decisioni capaci di far avanzare la lavorazione.
- Δt_k rappresenta il tempo simulato trascorso tra due decisioni consecutive. Questo termine introduce una penalizzazione proporzionale al tempo, con peso pari a 0.001, incentivando soluzioni più rapide senza rendere dominante il contributo temporale rispetto al progresso.
- S_k è una variabile binaria che assume valore 1 in assenza di progresso utile tra due decisioni consecutive, e 0 altrimenti. Il termine associato introduce una

penalità di stallo, con peso pari a 0.05, rendendo sfavorevoli le transizioni non produttive.

- T_k è una variabile binaria che assume valore 1 al completamento della ricetta, e 0 negli altri casi. È associata a un bonus terminale pari a 3.0, che rafforza il raggiungimento dell'obiettivo finale dell'episodio.

Il valore finale della reward viene inoltre limitato tramite clipping nell'intervallo [-3,+3]. Questa scelta evita che transizioni particolarmente favorevoli o sfavorevoli generino valori eccessivamente alti o bassi, che potrebbero influenzare troppo l'aggiornamento della politica. Il clipping mantiene quindi il segnale di apprendimento su una scala controllata durante l'addestramento.

4.3.2 Reward globale

La reward globale è progettata per guidare l'agente nella selezione della sequenza di ricette da eseguire, con l'obiettivo di ridurre il tempo complessivo di completamento dello scenario.

La funzione adottata è:

$$r_k = w_f \cdot \Delta F_k - w_t \cdot \Delta t_k$$

dove:

- ΔF_k rappresenta la variazione del numero di ricette ancora da completare tra due decisioni globali consecutive. Poiché nel modello una nuova decisione globale viene richiesta dopo l'esecuzione della ricetta selezionata, questo termine assume normalmente valore pari a 1. Il suo ruolo è quindi fornire un contributo positivo associato all'avanzamento dello scenario, mantenendo la reward coerente con il requisito di completamento. A tale termine è associato un peso pari a 0.1, scelto per introdurre questo contributo positivo senza renderlo dominante rispetto alla penalizzazione temporale.

- Δt_k rappresenta il tempo simulato trascorso per l'esecuzione della ricetta selezionata. Questo termine è associato a un peso pari a 0.01 e introduce una penalizzazione proporzionale alla durata dell'esecuzione. In questo modo, tra le scelte che portano al completamento, vengono favorite quelle che riducono il tempo complessivo del processo.

In questa configurazione, la reward globale mantiene la coerenza con i KPI definiti: il completamento delle ricette rimane il requisito da soddisfare, mentre il tempo simulato guida la scelta verso sequenze più efficienti.

4.4 Configurazione di addestramento

L'addestramento degli agenti è stato condotto separatamente per i due livelli decisionali del sistema. Questa separazione è coerente con l'architettura gerarchica adottata e consente di ottenere politiche specializzate per problemi decisionali diversi per frequenza, orizzonte temporale e significato delle azioni.

In entrambi i casi è stato utilizzato `MaskablePPO`, implementato nella libreria SB3-Contrib, poiché lo spazio delle azioni è discreto ma dipendente dallo stato corrente e richiede quindi il supporto all'action masking [24].

4.4.1 Architettura della politica

Per entrambi gli agenti è stata adottata una politica di tipo `MlpPolicy`. In SB3-Contrib, questa politica corrisponde a una `MaskableActorCriticPolicy`, cioè una struttura actor-critic compatibile con l'action masking [24].

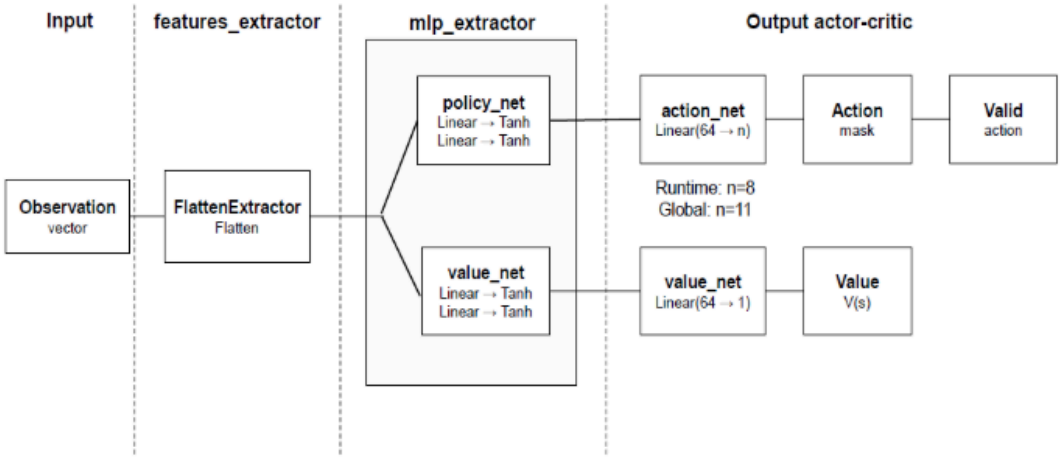


Figura 4 — Architettura della politica MaskablePPO utilizzata

La Figura 4 mostra l'architettura della politica utilizzata dai modelli addestrati. La rete riceve in ingresso un vettore di osservazione che è una rappresentazione numerica dello stato costruita dagli ambienti Gymnasium a partire dalle informazioni del Digital Model.

Il primo modulo della politica è il `FlattenExtractor`, il cui compito generale è convertire l'osservazione prodotta dall'ambiente in una rappresentazione vettoriale monodimensionale utilizzabile dalla rete neurale. Nel modello sviluppato, però, l'osservazione è già costruita dagli ambienti Gymnasium come un vettore numerico a dimensione fissa; di conseguenza, il `FlattenExtractor` non introduce una vera trasformazione informativa, ma si limita a inoltrare tale vettore alla rete successiva nel formato previsto da Stable-Baselines3 [30].

L'elaborazione principale è affidata al `MlpExtractor`, modulo che suddivide la rete in due rami fully-connected distinti: il ramo `policy_net`, associato all'actor, e il ramo `value_net`, associato al critic. Entrambi ricevono in ingresso il vettore prodotto dal `FlattenExtractor` e sono composti da due layer lineari da 64 neuroni ciascuno, con funzione di attivazione `Tanh` dopo ogni layer.

Il ramo `policy_net` produce una rappresentazione latente di 64 elementi, successivamente inviata al layer finale `action_net`. Nel modello operativo questo layer genera 8 logits, corrispondenti alle azioni discrete disponibili, mentre nel modello globale genera 11 logits, associati alle azioni previste per il livello globale. Prima della selezione dell'azione viene applicata la maschera delle azioni ammissibili, che esclude dalla distribuzione le azioni non valide nello stato corrente. I logits delle sole azioni valide vengono quindi trasformati in probabilità mediante softmax, costruendo la distribuzione da cui l'agente seleziona l'azione.

Parallelamente, il ramo `value_net` del `MlpExtractor` elabora lo stesso vettore di osservazione per stimare il valore dello stato corrente. La rappresentazione ottenuta viene inviata al layer finale del critic, che restituisce un singolo valore corrispondente alla stima $V(s)$, utilizzata da PPO per il calcolo del vantaggio e per l'aggiornamento dei parametri della politica [12].

4.4.2 Dataset di ricette e gestione degli ambienti di addestramento

Per l'addestramento e la valutazione degli agenti sono stati utilizzati insiemi distinti di dati, organizzati secondo la loro funzione: training per l'aggiornamento della politica, validation per il monitoraggio dell'addestramento e la selezione del miglior modello, test per il confronto finale con le baseline. La suddivisione non è stata impostata secondo una percentuale fissa, come avviene spesso nei dataset statici, perché nel Reinforcement Learning l'esperienza viene generata durante l'interazione tra agente e ambiente. Una singola ricetta o uno scenario multi-ricetta può infatti produrre più punti decisionali e quindi più transizioni utilizzate per aggiornare la politica [7]. Di conseguenza, il numero di ricette o scenari non coincide direttamente con il numero effettivo di esempi osservati dall'agente durante l'addestramento.

Nel modello sviluppato, i dati sono stati organizzati in modo diverso per i due ambienti di Reinforcement Learning. Per l'ambiente operativo, l'unità di riferimento è la singola ricetta: sono state utilizzate 1600 ricette per il training, 200 ricette per la validation e 200 ricette per il test.

Per l'ambiente globale, invece, l'unità di riferimento è lo scenario multi-ricetta, poiché l'agente determina la sequenza di esecuzione tra più ricette disponibili. Il modello globale è stato addestrato su 300 scenari di training, ciascuno composto da 10 ricette, per un totale di 3000 ricette incluse negli scenari di addestramento. La validation è stata effettuata su 180 scenari della stessa struttura, mentre la valutazione finale del livello globale e della configurazione completa è stata condotta su ulteriori 180 scenari di test, corrispondenti a 1800 ricette processate nel confronto finale.

Gli ambienti di addestramento sono stati gestiti tramite DummyVecEnv di Stable-Baselines3. Questo componente consente di creare più istanze indipendenti dello stesso ambiente Gymnasium e di esporle all'algoritmo PPO attraverso un'unica interfaccia vettoriale [32]. Tale interfaccia restituisce, a ogni passo di simulazione, un insieme di valori: un'osservazione per ciascun ambiente, una ricompensa per ciascun ambiente e un'indicazione di terminazione per ciascun ambiente.

L'utilizzo di ambienti vettorializzati permette quindi a PPO di raccogliere dati da più simulazioni indipendenti durante l'addestramento. Nel caso considerato, questo è utile perché le diverse istanze possono evolvere secondo ricette, stati e traiettorie differenti, aumentando la varietà delle esperienze osservate dalla politica durante la fase di apprendimento.

4.4.3 Normalizzazione e iperparametri

Alla normalizzazione già effettuata all'interno degli ambienti Gymnasium, che trasformano le informazioni del Digital Model in feature numeriche scalate e coerenti con lo spazio di osservazione, si aggiunge durante l'addestramento l'utilizzo di `VecNormalize`. Questo strumento mantiene statistiche progressive sulle osservazioni, in particolare media e varianza, e le utilizza per normalizzare gli input forniti alla rete durante il training [32].

Nel modello sviluppato, `VecNormalize` è stato applicato esclusivamente alle osservazioni, perché queste rappresentano lo stato utilizzato dalla politica per scegliere

le azioni. Anche se le feature sono già scalate negli ambienti Gymnasium, la loro distribuzione può variare durante l'addestramento; la normalizzazione contribuisce quindi a stabilizzare l'apprendimento. La reward, invece, è stata mantenuta nella sua scala originale, poiché costituisce il segnale con cui viene valutata l'azione scelta ed è stata progettata esplicitamente a partire dai KPI del sistema. Una sua normalizzazione automatica avrebbe modificato il significato dei pesi assegnati ai diversi termini della funzione di ricompensa, rendendo meno diretto il collegamento tra reward e prestazioni effettive del sistema.

Per l'agente operativo sono stati utilizzati 3.000.000 timestep di addestramento, mentre per l'agente globale sono stati utilizzati 500.000 timestep. In PPO, un timestep corrisponde a un'interazione tra agente e ambiente; nel modello sviluppato questa interazione avviene quando il sistema raggiunge un punto decisionale.

Il numero di timestep è stato differenziato in base alla natura dei due livelli decisionali. Il livello operativo richiede un addestramento più lungo perché interviene molte volte durante l'esecuzione di una ricetta, prendendo decisioni locali frequenti sulla tipologia da assegnare ai mover. Il livello globale, invece, prende decisioni meno frequenti, legate alla selezione della prossima ricetta da eseguire, e opera quindi su una scala temporale più ampia. Per questo motivo è stato addestrato con un numero inferiore di timestep.

Gli altri iperparametri definiscono il modo in cui PPO raccoglie le esperienze e aggiorna la rete neurale secondo la configurazione prevista dall'implementazione di Stable-Baselines3 [31]. Il parametro `n_steps` stabilisce quante decisioni vengono raccolte da ciascun ambiente prima di eseguire un aggiornamento della politica. È stato impostato a 256 per l'agente operativo e a 1024 per l'agente globale. Nel primo caso, un valore più basso consente aggiornamenti più frequenti, coerenti con la natura locale e ripetuta delle decisioni operative. Nel secondo caso, un valore più alto permette di raccogliere più transizioni prima dell'aggiornamento, ottenendo una stima più stabile dell'effetto delle scelte globali.

La batch_size, pari a 256 per entrambi gli agenti, indica quante transizioni vengono usate insieme in ogni mini-batch durante l'ottimizzazione. Questo valore permette di

aggiornare la politica su gruppi di dati sufficientemente consistenti, mantenendo l'ottimizzazione gestibile.

Il parametro `n_epochs`, pari a 6, indica quante volte PPO riutilizza lo stesso insieme di transizioni raccolte prima di acquisire nuovi dati dagli ambienti. A ogni epoca, le transizioni vengono rimescolate e suddivise in mini-batch per aggiornare la politica e la funzione di valore. In questo modo le esperienze disponibili vengono sfruttate meglio senza un riutilizzo eccessivo degli stessi dati.

Il `learning_rate` controlla quanto vengono modificati i parametri della rete a ogni aggiornamento. Per entrambi gli agenti sono stati utilizzati valori contenuti: $2{,}5 \cdot 10^{-4}$ per l'agente operativo e $3 \cdot 10^{-4}$ per l'agente globale. Questa scelta consente aggiornamenti graduali della politica, riducendo il rischio di variazioni troppo brusche durante l'apprendimento.

Gli iperparametri `gamma` e `gae_lambda` intervengono nella stima del vantaggio $\hat{A}_t$, utilizzata da PPO per valutare se un'azione ha prodotto un risultato migliore o peggiore rispetto a quanto previsto dal critic. Questi parametri regolano quanto la valutazione di una decisione debba dipendere dal reward immediato e quanto, invece, debba considerare anche gli effetti osservati nelle transizioni successive.

Il valore di `gamma` indica quanto peso dare alle ricompense future rispetto a quella immediata ed è stato impostato a 0.995 per entrambi gli agenti, così da mantenere rilevanti gli effetti successivi delle decisioni. Il parametro `gae_lambda`, pari a 0.95, regola quanto la stima del vantaggio consideri le transizioni future, dando peso progressivamente minore a quelle più lontane. Entrambi i valori sono coerenti con il modello sviluppato, in cui una decisione può influenzare non solo lo stato successivo, ma anche il tempo complessivo di completamento della ricetta o della sequenza di ricette.

Il parametro `clip_range` controlla quanto PPO può modificare la politica a ogni aggiornamento, limitando variazioni troppo brusche nelle probabilità assegnate alle azioni rispetto alla politica precedente. È stato impostato a 0.15 per l'agente operativo e a 0.3 per l'agente globale. Nel livello operativo, dove le decisioni sono numerose e

ravvicinate durante l'esecuzione della ricetta, un valore più basso rende gli aggiornamenti più graduali e riduce il rischio che la politica cambi eccessivamente a causa di variazioni locali del segnale di apprendimento. Nel livello globale le decisioni sono meno frequenti e riguardano la selezione delle ricette; per questo è stato adottato un valore più alto, che consente alla politica di adattarsi con maggiore flessibilità quando emergono differenze significative tra diverse strategie di sequenziamento.

Il coefficiente di entropia, passato all'algoritmo tramite `ent_coef`, controlla il peso del termine che incentiva l'esplorazione della politica. In pratica, evita che l'agente concentri troppo presto la probabilità su poche azioni considerate migliori. È stato impostato a 0.008 per l'agente operativo e a 0.01 per quello globale. Entrambi sono valori contenuti, scelti per mantenere una componente di esplorazione senza rendere le decisioni eccessivamente casuali. Il valore leggermente più alto nel livello globale favorisce il confronto tra diverse sequenze di ricette, mentre quello più basso nel livello operativo limita la casualità nelle decisioni locali, più frequenti durante l'esecuzione della ricetta.

Il coefficiente della funzione di valore, passato all'algoritmo tramite `vf_coef`, stabilisce quanto peso assegnare all'errore del critic nella stima del valore dello stato $V(s)$. Questa stima è importante perché PPO la utilizza per calcolare il vantaggio $\hat{A}_t$, cioè per valutare se un'azione è stata migliore o peggiore rispetto a quanto atteso. Se il critic stima male il valore dello stato, anche il vantaggio diventa meno affidabile e l'actor può aggiornare la politica in modo non corretto. Nel progetto `vf_coef` è stato impostato a 0.7 per l'agente operativo e a 0.8 per quello globale, assegnando quindi un peso significativo alla qualità della stima del critic.

4.5 Monitoraggio dell'addestramento e selezione dei modelli

4.5.1 Valutazione periodica e scelta del modello migliore

Stable-Baselines3 mette a disposizione una callback di valutazione, `EvalCallback`, che consente di testare periodicamente il modello durante l'addestramento su un ambiente separato. A intervalli prefissati, la callback esegue alcuni episodi di valutazione, calcola la prestazione media ottenuta e, se questa è migliore rispetto alle valutazioni precedenti, la callback salva automaticamente il modello e i parametri di normalizzazione associati [33].

Nel progetto, tuttavia, non è stata utilizzata direttamente la `EvalCallback` standard, perché il criterio di selezione del modello doveva essere legato al KPI principale del sistema, cioè il tempo simulato di completamento, e non alla sola reward media restituita dall'ambiente. Per questo motivo è stata implementata una callback personalizzata, basata sul meccanismo di callback di Stable-Baselines3, ma adattata alla logica del Digital Model.

La callback sviluppata controlla il numero di timesteps eseguiti e, a intervalli prefissati, valuta il modello corrente sulle ricette del validation set. Durante questa fase, la politica viene eseguita rispettando la maschera delle azioni ammissibili e, al termine di ogni episodio, viene registrato il tempo simulato necessario al completamento. Il punteggio assegnato all'episodio è definito come l'opposto di tale tempo: in questo modo, un tempo più basso produce un punteggio più alto. La callback calcola quindi il punteggio medio sulle ricette di validazione e lo confronta con il miglior valore ottenuto fino a quel momento. Se il nuovo valore è migliore, il modello corrente viene salvato come migliore modello, insieme alle statistiche di normalizzazione necessarie per riutilizzarlo correttamente nelle successive fasi di valutazione.

4.5.2 Metriche di addestramento e risultati osservati

L'andamento dell'addestramento è stato analizzato attraverso i log prodotti da Stable-Baselines3, considerando sia metriche prestazionali sia metriche interne di PPO. Le prime descrivono il comportamento dell'agente nell'ambiente, mentre le seconde permettono di valutare la stabilità dell'ottimizzazione e la qualità della funzione di valore appresa.

La `ep_rew_mean`, ossia la reward media per episodio, descrive il ritorno medio ottenuto dall'agente durante il training. In entrambi i livelli si osserva un miglioramento, segno che le politiche apprendono progressivamente strategie più efficaci. Nel livello operativo la reward passa da valori intorno a 4 a valori prossimi a 5, mentre a livello globale mostra un miglioramento più contenuto, passando da circa -13,2 a circa -12,6. L'andamento conferma quindi che, durante l'addestramento, entrambi gli agenti migliorano la qualità delle decisioni rispetto alle prime fasi del training.

La `ep_len_mean`, cioè la lunghezza media degli episodi, indica il numero medio di decisioni eseguite dall'agente in un episodio. Questa metrica è significativa soprattutto per il livello operativo, dove il numero di decisioni dipende da come l'agente gestisce il riempimento della ricetta. Nei log si osserva una diminuzione da circa 61 a circa 46 decisioni, segno che la politica impara progressivamente a completare la ricetta con meno scelte operative. Nel livello globale questa metrica non è particolarmente rilevante, poiché il numero di decisioni dipende dal numero fissato di ricette considerate nello scenario, e non direttamente dall'efficienza della politica.

Le metriche interne di PPO permettono di verificare che l'aggiornamento della politica avvenga in modo stabile durante l'addestramento [34]. In particolare, `approx_kl` misura la differenza tra la politica prima e dopo l'aggiornamento: valori contenuti indicano che la nuova politica non si discosta eccessivamente dalla precedente. Nei log analizzati questa metrica rimane bassa, con valori finali circa pari a 0,001 nel livello operativo e a 0,013 nel livello globale, indicando aggiornamenti graduali e controllati. La `clip_fraction` indica quanto spesso PPO applica il meccanismo di clipping per limitare modifiche troppo ampie della politica. Anche questa metrica rimane su valori moderati, circa 1–2% nel livello operativo e circa 4% nel livello globale. Nel

complesso, questi risultati indicano che PPO aggiorna la politica in modo progressivo: il clipping interviene solo in una parte limitata degli aggiornamenti e non emergono variazioni improvvise che possano compromettere la stabilità dell'apprendimento.

La metrica `entropy_loss` è legata all'entropia della politica, cioè al grado di esplorazione mantenuto dall'agente nella scelta delle azioni. Un'entropia più alta indica una distribuzione più ampia delle probabilità tra le azioni disponibili, mentre una sua riduzione indica che la politica tende a concentrarsi su un numero più limitato di azioni. Nei log analizzati, l'entropy_loss si avvicina a zero in entrambi i livelli: nel modello operativo passa da circa -0,94 a circa -0,15, mentre nel modello globale da circa -1,51 a circa -0,74. Questo andamento mostra che, durante l'addestramento, la politica diventa progressivamente meno esplorativa e più definita nelle proprie scelte.

Le metriche `explained_variance` e `value_loss` permettono di valutare la qualità del critic, cioè della rete che stima il valore degli stati. L'`explained_variance` misura quanto la stima del critic sia coerente con i ritorni osservati: valori più alti indicano una stima più accurata. Nei log cresce fino a circa 0,81 nel livello operativo, con picchi superiori a 0,87, e raggiunge valori prossimi a 0,99 nel livello globale. La stessa tendenza è confermata dalla `value_loss`, che rappresenta l'errore della funzione di valore: nel livello operativo si riduce da valori superiori a 2 fino a circa 0,14, mentre nel livello globale passa da circa 40 a circa 0,07. Nel complesso, questi andamenti indicano che il critic migliora progressivamente la propria capacità di valutare gli stati durante l'addestramento.

Nel complesso, i log indicano un addestramento regolare: le politiche migliorano progressivamente e le metriche interne di PPO non evidenziano instabilità significative nel processo di ottimizzazione.

4.6 Test sperimentali e risultati ottenuti

I test sono stati organizzati su tre livelli: valutazione della sola politica operativa, valutazione della sola politica globale e valutazione del sistema completo, nel quale sia la selezione della ricetta sia le decisioni operative sono affidate a politiche apprese tramite Reinforcement Learning. Questa organizzazione consente di analizzare separatamente il contributo del livello operativo, quello del livello globale e l'effetto combinato dell'architettura gerarchica completa.

In tutti i casi, le politiche apprese sono state confrontate con strategie euristiche di riferimento utilizzando i modelli migliori salvati durante l'addestramento. In fase di test sono stati inoltre caricati i parametri di `VecNormalize`, necessari per trasformare le osservazioni secondo la stessa scala usata nel training. Durante la valutazione tali statistiche non vengono aggiornate, ma soltanto applicate agli scenari di test [32].

La reward non è stata normalizzata, coerentemente con la configurazione adottata durante l'addestramento. In questo modo il valore restituito dall'ambiente mantiene un collegamento diretto con i KPI definiti per la valutazione.

4.6.1 Politiche euristiche utilizzate per il confronto

Per valutare l'efficacia delle politiche apprese tramite Reinforcement Learning, sono state utilizzate diverse politiche euristiche di riferimento. Queste baseline non vengono addestrate, ma selezionano l'azione attraverso regole deterministiche definite a priori. Il loro scopo è fornire un termine di confronto stabile e interpretabile rispetto alle politiche ottenute tramite addestramento.

Le baseline sono state adattate ai due livelli decisionali del sistema. A livello operativo, l'azione corrisponde alla scelta della tipologia di oggetto da assegnare al mover attivo; a livello globale, invece, corrisponde alla selezione della prossima ricetta da eseguire. Queste strategie mantengono quindi una logica simile nei due livelli, ma vengono applicate a oggetti decisionali diversi.

La politica sequential rappresenta la strategia più semplice. Essa seleziona la prima azione valida secondo un ordine prefissato: nel livello operativo corrisponde alla prima tipologia disponibile, mentre nel livello globale alla prima ricetta non ancora completata.

La politica greedy adotta un criterio basato sull'utilità immediata dell'azione. Nel livello operativo sceglie la tipologia disponibile più utile nello stato corrente, calcolando un punteggio che combina il fabbisogno residuo della ricetta attiva e quello del contenitore associato al mover. Poiché il fabbisogno della ricetta ha peso maggiore, la scelta non dipende solo dalla richiesta locale del mover, ma privilegia le tipologie che contribuiscono maggiormente al completamento complessivo della lavorazione. Nel livello globale, questa logica seleziona la ricetta più compatibile con la configurazione corrente degli alimentatori vibranti, con l'obiettivo di ridurre cambi di tipologia e tempi di riconfigurazione.

La politica spt_like è ispirata alla regola Shortest Processing Time [35]. Tale regola privilegia le lavorazioni più brevi, dando priorità alle attività meno onerose. Nel livello operativo, questa logica porta a scegliere la tipologia disponibile con il carico residuo più basso, considerando sia il fabbisogno del contenitore associato al mover sia quello della ricetta attiva. Nel livello globale, invece, viene selezionata la ricetta con il carico di lavoro residuo minore.

Tutte le baseline rispettano l'insieme delle azioni ammissibili nello stato corrente e utilizzano solo informazioni disponibili al momento della decisione. Il confronto con le politiche RL risulta quindi coerente: le euristiche applicano regole deterministiche definite manualmente, mentre gli agenti RL selezionano le azioni tramite una politica appresa dall'interazione con il Digital Model.

4.6.2 Valutazione della politica operativa

Il primo gruppo di test ha riguardato la valutazione della politica operativa misurando l'efficacia dell'agente RL nelle decisioni locali, confrontandolo con le tre baseline operative descritte in precedenza.

La valutazione è stata condotta sul test set operativo. Per ciascuna delle 200 ricette di test, la politica appresa è stata confrontata con ognuna delle tre baseline operative, ottenendo complessivamente 600 confronti diretti. Tutte le politiche considerate hanno completato correttamente gli scenari valutati. Di conseguenza, il confronto sui tempi di completamento è valido in tutti i 600 confronti.

Nel complesso, la politica RL ha ottenuto un tempo medio pari a 702,04 unità di tempo simulato, contro 878,27 delle baseline. La riduzione media è quindi pari a 176,23 unità di tempo simulato e corrisponde a un miglioramento medio del 18,47%, calcolato sui singoli confronti validi. Come si vede dalla Tabella 1 il vantaggio risulta particolarmente evidente rispetto alle baseline greedy e spt_like, mentre il confronto con sequential è meno uniforme, pur rimanendo favorevole alla politica RL in termini di tempo medio.

Baseline	**Confronti validi**	**Tempo medio RL**	**Tempo medio baseline**	**Riduzione media tempo**	**Miglioramento medio**	**Vittorie RL**
greedy	200	702,04	893,75	191,70	20,79%	200/200
sequential	200	702,04	849,34	147,29	15,07%	182/200
spt	200	702,04	891,73	189,69	19,55%	200/200
Totale	**600**	**702,04**	**878,27**	**176,23**	**18,47%**	**582/600**

Tabella 1 - Risultati del confronto tra la politica operativa RL e le baseline sul test set.

4.6.3 Valutazione della politica globale

Il secondo gruppo di test ha riguardato la valutazione della politica globale, incaricata di scegliere la prossima ricetta da eseguire tra quelle ancora disponibili nello scenario.

L’obiettivo è verificare se la politica appresa sia in grado di ordinare le ricette in modo più efficace rispetto alle strategie euristiche globali.

Per rendere il confronto centrato sulla sola decisione globale, il criterio decisionale utilizzato a livello operativo è stato mantenuto invariato in tutte le prove. Dopo la selezione della ricetta, quindi, l’esecuzione locale è stata gestita sempre con lo stesso criterio, mentre a cambiare è stata soltanto la strategia di scelta della ricetta successiva. La politica globale appresa è stata confrontata con le tre baseline globali descritte in precedenza, per un totale di 540 confronti.

La Tabella 2 mostra come la politica globale RL completi correttamente tutti gli episodi valutati. Il tempo medio ottenuto è pari a 5642,91 unità di tempo simulato, contro 5656,45 delle baseline globali. Il confronto risulta quindi molto equilibrato: su 540 confronti complessivi, la politica RL ottiene un tempo inferiore in 306 casi, mentre le baseline risultano migliori nei restanti 234. Il miglioramento medio complessivo è positivo, ma molto contenuto, pari allo 0,16%.

Baseline	**Confronti validi**	**Tempo medio RL**	**Tempo medio baseline**	**Riduzione media tempo**	**Miglioramento medio**	**Vittorie RL**
sequential	180	5642,91	5650,43	7,52	0,03%	91/180
spt	180	5642,91	5669,31	26,40	0,39%	117/180
greedy	180	5642,91	5649,63	6,72	0,05%	98/180
Totale	**540**	**5642,91**	**5656,45**	**13,54**	**0,16%**	**306/540**

Tabella 2 – Risultati del confronto tra la politica globale RL e le baseline sul test set.

4.6.4 Valutazione dell'architettura gerarchica completa

Il terzo gruppo di test ha riguardato la valutazione dell'architettura gerarchica completa, nella quale vengono utilizzate insieme la politica globale RL e la politica operativa RL. Questa configurazione è stata confrontata con le combinazioni euristiche ottenute associando le baseline globali e operative, per un totale di 1620 confronti complessivi.

La configurazione completa basata su RL porta a termine correttamente tutti i confronti. Le combinazioni euristiche, invece, completano correttamente 288 confronti su 1620; nei restanti casi non raggiungono il completamento entro i limiti previsti dalla simulazione. Questo risultato evidenzia che, nella valutazione completa, la differenza tra RL e baseline non riguarda solo il tempo di esecuzione, ma anche la capacità di completare stabilmente gli scenari.

Per questo motivo, il confronto sui tempi è stato calcolato solo nei 288 casi in cui anche la combinazione euristica completa correttamente l'episodio. Nei casi in cui RL completa e la baseline non completa, il confronto è stato considerato favorevole alla configurazione RL, ma non è stato utilizzato per calcolare il miglioramento percentuale sul tempo. Infatti, il tempo associato a un episodio non completato non rappresenta un vero tempo di completamento.

La Tabella 3 mostra che, nei confronti validi nei quali la configurazione RL risulta essere sempre la migliore, essa ottiene un tempo medio pari a 5233,99 unità di tempo simulato, mentre le baseline ottengono un tempo medio pari a 6651,28. Il miglioramento medio sul tempo è quindi pari al 21,09% e la riduzione media è pari a 1417,29 unità di tempo simulato.

Baseline glob/loc	Confronti validi	Tempo medio RL	Tempo medio baseline	Riduzione media tempo	Miglioramento medio
seq / seq	54	5231,95	6384,13	1152,18	18,05%
seq / greedy	9	5198,40	6962,36	1763,96	25,34%
seq / spt	36	5568,61	7379,55	1810,94	24,54%
spt / seq	34	4850,65	5817,94	967,29	16,63%
spt / greedy	11	5179,40	6921,40	1742,00	25,17%
spt / spt	45	5061,62	6551,36	1489,74	22,74%
greedy / seq	54	5214,43	6392,77	1178,34	18,43%
greedy / greedy	10	5183,30	6977,89	1794,59	25,72%
greedy / spt	35	5557,96	7393,04	1835,08	24,82%
Totale	**288**	**5233,99**	**6651,28**	**1417,29**	**21,09%**

Tabella 3 – Risultati del confronto tra l'architettura completa RL e le combinazioni di baseline.

4.7 Considerazioni conclusive

Nel complesso, i risultati sperimentali confermano il raggiungimento dell'obiettivo della tesi. Le politiche apprese tramite Reinforcement Learning hanno contribuito all'ottimizzazione del processo produttivo simulato, riducendo i tempi di completamento delle ricette rispetto alle strategie euristiche considerate. Il Digital

Model del processo produttivo ha reso possibile eseguire l'addestramento e la valutazione delle politiche in condizioni controllate e riproducibili, permettendo di analizzare separatamente il livello operativo, il livello globale e la configurazione completa.

Il contributo più evidente emerge nel livello operativo. In questo caso, la politica RL riduce in modo significativo il tempo medio di completamento rispetto alle baseline, confermando la capacità dell'agente di apprendere decisioni locali più efficaci nella scelta delle tipologie da assegnare ai mover. Questo risultato indica che il Reinforcement Learning risulta particolarmente efficace nelle decisioni frequenti e direttamente collegate all'avanzamento della lavorazione.

Il livello globale mostra invece un miglioramento medio molto più contenuto. La politica appresa risulta competitiva rispetto alle strategie euristiche, ma non nettamente superiore. Questo suggerisce che la sola scelta della sequenza di ricette ha un impatto meno marcato sulle prestazioni complessive, anche perché il risultato finale dipende in larga parte dalle decisioni operative eseguite durante la lavorazione.

Il risultato più rilevante si osserva nella configurazione completa. L'integrazione tra politica globale e politica operativa consente di completare tutti gli scenari valutati e di ottenere tempi inferiori nei confronti validi. Questo conferma l'utilità dell'architettura gerarchica proposta, nella quale i due livelli decisionali contribuiscono in modo complementare alla gestione del sistema.

I risultati ottenuti mostrano quindi che l'approccio sviluppato costituisce una base efficace per sperimentare politiche decisionali apprese e confrontarle con strategie euristiche. Allo stesso tempo, rappresenta un punto di partenza per attività successive, quali la calibrazione del modello, la validazione su dati reali e una possibile evoluzione verso applicazioni più vicine al sistema produttivo effettivo.

Bibliografia e sitografia

[1] Y. Di, L. Deng, L. Wang e L. Zhang, “A Review of Deep Reinforcement Learning Methods for Production Scheduling,” *Tsinghua Science and Technology*, vol. 31, n. 5, pp. 2504–2533, 2025. DOI: 10.26599/TST.2024.9010247.

[2] Beckhoff Automation GmbH & Co. KG, “XPlanar | Planar motor system.” [Online]. Disponibile: https://www.beckhoff.com/en-en/products/motion/xplanar-planar-motor-system/. [Consultato: 9 aprile 2026].

[3] W. Kritzinger, M. Karner, G. Traar, J. Henjes e W. Sihn, “Digital Twin in manufacturing: A categorical literature review and classification,” *IFAC-PapersOnLine*, vol. 51, n. 11, pp. 1016–1022, 2018. DOI: 10.1016/j.ifacol.2018.08.474.

[4] C. G. Cassandras e S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. New York, NY, USA: Springer, 2008.

[5] R. S. Sutton, D. Precup e S. Singh, “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning,” *Artificial Intelligence*, vol. 112, n. 1–2, pp. 181–211, 1999. DOI: 10.1016/S0004-3702(99)00052-1.

[6] W. B. Powell, “Sequential Decision Analytics and Modeling: Modeling with Python - Part I,” *Foundations and Trends in Technology, Information and Operations Management*, vol. 15, n. 4, pp. 325–456, 2022. DOI: 10.1561/0200000103-I.

[7] R. S. Sutton e A. G. Barto, *Reinforcement Learning: An Introduction, 2ª ed. Cambridge*, MA, USA: MIT Press, 2018.

[8] C. J. C. H. Watkins e P. Dayan, “Q-learning,” *Machine Learning*, vol. 8, pp. 279–292, 1992. DOI: 10.1007/BF00992698.

[9] V. Mnih et al., “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, n. 7540, pp. 529–533, 2015. DOI: 10.1038/nature14236.

[10] G. Kalweit, M. Huegle, M. Werling e J. Boedecker, “Deep Constrained Q-learning,” arXiv preprint arXiv:2003.09398, 2020.

[11] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” Machine Learning, vol. 8, pp. 229–256, 1992. DOI: 10.1007/BF00992696.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford e O. Klimov, “Proximal Policy Optimization Algorithms,” arXiv preprint arXiv:1707.06347, 2017.

[13] H.-K. Lim, J.-B. Kim, J.-S. Heo e Y.-H. Han, “Federated Reinforcement Learning for Training Control Policies on Multiple IoT Devices,” Sensors, vol. 20, n. 5, art. 1359, 2020. DOI: 10.3390/s20051359.

[14] S. Huang e S. Ontañón, “A Closer Look at Invalid Action Masking in Policy Gradient Algorithms,” in Proceedings of the Thirty-Fifth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2022, Jensen Beach, FL, USA, 2022. arXiv preprint arXiv:2006.14171.

[15] T. G. Dietterich, “Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition,” Journal of Artificial Intelligence Research, vol. 13, pp. 227–303, 2000.

[16] AnyLogic, “Reinforcement Learning experiment — AnyLogic Help,” *AnyLogic Documentation*. [Online]. Disponibile: https://anylogic.help/anylogic/experiments/rl-experiment.html. [Consultato: 12 maggio 2026].

[17] Rockwell Automation, “Discrete Event Modeling | Arena Simulation Software,” *Rockwell Automation*. [Online]. Disponibile: https://www.rockwellautomation.com/en-us/products/software/arena-simulation/discrete-event-modeling.html. [Consultato: 12 maggio 2026].

[18] SimPy Developers, “Overview — SimPy Documentation,” *SimPy 4.1 Documentation*. [Online]. Disponibile: https://simpy.readthedocs.io/en/latest/. [Consultato: 12 maggio 2026].

[19] SimPy Developers, “SimPy basics — How SimPy works,” *SimPy 4.1 Documentation*. [Online]. Disponibile: https://simpy.readthedocs.io/en/latest/topical_guides/simpy_basics.html. [Consultato: 12 maggio 2026].

[20] SimPy Developers, “Shared Resources — SimPy Documentation,” *SimPy 4.1 Documentation*. [Online]. Disponibile: https://simpy.readthedocs.io/en/latest/topical_guides/resources.html. [Consultato: 12 maggio 2026].

[21] SimPy Developers, “Events — SimPy Documentation,” *SimPy 4.1 Documentation*. [Online]. Disponibile: https://simpy.readthedocs.io/en/latest/topical_guides/events.html. [Consultato: 12 maggio 2026].

[22] SimPy Developers, “Environments — SimPy Documentation,” *SimPy 4.1 Documentation*. [Online]. Disponibile: https://simpy.readthedocs.io/en/latest/topical_guides/environments.html. [Consultato: 12 maggio 2026].

[23] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus e N. Dormann, “Stable-Baselines3: Reliable Reinforcement Learning Implementations,” Journal of Machine Learning Research, vol. 22, n. 268, pp. 1–8, 2021. [Online]. Disponibile: https://jmlr.org/papers/v22/20-1364.html.

[24] Stable-Baselines-Team, “Maskable PPO,” Stable-Baselines3 Contrib Documentation. [Online]. Disponibile: https://sb3-contrib.readthedocs.io/en/master/modules/ppo_mask.html. [Consultato: 17 maggio 2026].

[25] M. Towers et al., “Gymnasium: A Standard Interface for Reinforcement Learning Environments,” arXiv preprint arXiv:2407.17032, 2024. [Online]. Disponibile: https://arxiv.org/abs/2407.17032.

[26] Farama Foundation, “Env — Gymnasium Documentation,” Gymnasium Documentation. [Online]. Disponibile: https://gymnasium.farama.org/api/env/. [Consultato: 12 maggio 2026].

[27] Farama Foundation, “Spaces — Gymnasium Documentation,” Gymnasium Documentation. [Online]. Disponibile: https://gymnasium.farama.org/api/spaces/. [Consultato: 12 maggio 2026].

[28] R. G. Sargent, “Verification and validation of simulation models,” in Proceedings of the 2010 Winter Simulation Conference, Baltimore, MD, USA, 2010, pp. 166–183. DOI: 10.1109/WSC.2010.5679166.

[29] T. Akiba, S. Sano, T. Yanase, T. Ohta e M. Koyama, “Optuna: A next-generation hyperparameter optimization framework,” in Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Anchorage, AK, USA, 2019, pp. 2623–2631. DOI: 10.1145/3292500.3330701.

[30] Stable-Baselines3 Developers, “Policy Networks,” *Stable-Baselines3 Documentation*. [Online]. Disponibile: https://stable-baselines3.readthedocs.io/en/master/guide/custom_policy.html. [Consultato: 20 maggio 2026].

[31] Stable-Baselines3 Developers, “PPO,” *Stable-Baselines3 Documentation*. [Online]. Disponibile: https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html. [Consultato: 22 maggio 2026].

[32] Stable-Baselines3 Developers, “Vectorized Environments,” *Stable-Baselines3 Documentation*. [Online]. Disponibile: https://stable-baselines3.readthedocs.io/en/master/guide/vec_envs.html. [Consultato: 22 maggio 2026].

[33] Stable-Baselines3 Developers, “Callbacks — Stable-Baselines3 Documentation”, sezione *EvalCallback*. [Online]. Disponibile: https://stable-baselines3.readthedocs.io/en/master/guide/callbacks.html. [Consultato: 22 maggio 2026].

[34] Stable-Baselines3 Developers, “Logger — Stable-Baselines3 Documentation”, sezione *Explanation of logger output*. [Online]. Disponibile: https://stable-baselines3.readthedocs.io/en/master/common/logger.html. [Consultato: 22 maggio 2026].

[35] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 3rd ed. New York, NY, USA: Springer, 2008, pp. 130–131.