



UNIVERSITÀ DEGLI STUDI DI CATANIA

DIPARTIMENTO DI INGEGNERIA ELETTRICA
ELETTRONICA E INFORMATICA

Corso di Laurea Magistrale in Ingegneria Informatica

Quantization and Training of DNNs for Efficient MAC Protocol Learning in Industrial IoT Networks

Relatore:

Prof. Salvatore Riolo

Candidato:

Antonino Salemi

Correlatori:

Prof. Luciano Miuccio
Prof.ssa Daniela Panno

Sommario

The Industrial Internet of Things (IIoT) represents a convergence of advanced manufacturing processes and cutting-edge information technology and is expected to be among the driving forces of global economic growth in the coming decade. A key aspect of IIoT development is the interconnection of IIoT devices (IIoTDs), which facilitate improved monitoring, automation, and data analytics within industrial environments.

To enable efficient communication among IIoTDs, the integration of Artificial Intelligence (AI) can enhance the customizability and adaptability of the communication process, particularly at the Medium Access Control (MAC) layer. The implementation of Multi-Agent Deep Reinforcement Learning (MADRL) solutions for MAC protocol learning in industrial environments has gained increasing attention, thus requiring the integration of Deep Neural Networks (DNNs) within IIoTDs. However, these devices often operate under critical conditions due to severe constraints on computational capacity, storage, and energy. Consequently, deploying DNNs on resource-constrained IoT devices presents significant challenges, making the development of lightweight and efficient solutions essential to maintaining performance without significant degradation.

This thesis analyzes model quantization techniques aimed at reducing the complexity of DNNs while maintaining acceptable performance in the performances of MADRL-based MAC protocol. First, the Post-Training Quantization (PTQ) techniques (i.e., static quantization and dynamic quantization) have been analyzed, where a trained model is converted from floating-point precision to a quantized representation after training. Second, Quantization-Aware Training (QAT) has been applied, in which training is performed using a simulated reduced-precision model to help the network in adapting to quantization.

For each of these techniques, the impact on several DNN-based MAC protocols is evaluated, with a focus on the effects of weight and activation precision reduction. In particular, for the PTQ static technique, the influence of the calibration set size on the precision of the DNNs' output distribution has been analyzed, where the precision depends on both the number of points in the calibration set and the memory length of the agent's observation. Additionally, the impact of the calibration set size varying the memory length is analyzed with respect to the communication Key Performance Indicators (KPIs).

Considering the Python environment as the coding library for all quantization techniques, the analyses were conducted using the following configurations. For dynamic

quantization, the default configuration was considered. For static quantization, both the default configurations, where weights and activations are quantized using different observers, and a set of custom configurations were explored.

Finally, for QAT, both the default configuration, which applies the same training parameters as in the MADRL model, and an improved QAT version with custom fine-tuned parameters are implemented.

Through various simulations, the different quantization techniques are compared in terms of memory footprint, runtime memory usage, and communication KPI performance. The analysis indicates that, in the context of MAC protocols, the most suitable quantization technique depends on the available deployment time, i.e., the time before the system needs to be set up. If deployment time is limited, dynamic quantization represents the best choice, followed by static quantization and, lastly, QAT. However, when sufficient deployment time is available, QAT is recommended, as it can provide performance close to or even surpass the original model when properly configured and fine-tuned.

Indice

1 Reti 6G ed Industrial Internet of Things	1
1.1 Storia delle reti mobili	1
1.2 Introduzione al 6G	3
1.3 Scenari d'uso 6G	5
1.4 Tecnologie abilitanti	7
1.5 Architettura della rete 6G	10
1.5.1 Core Network	11
1.5.2 Radio Access Network	13
1.6 Intelligenza artificiale nella rete 6G	16
2 Multi-agent deep reinforcement learning	18
2.1 Reinforcement Learning	18
2.1.1 Algoritmi di Reinforcement Learning	21
2.2 Algoritmi MARL	25
2.2.1 Fully Centralized Learning	29
2.2.2 Indipendent Learning	30
2.2.3 CTDE	31
2.3 Algoritmi MADRL	32
2.3.1 Deep Q-Network	33
2.3.2 Multi-Agent Proximal Policy Optimization	33
3 Quantizzazione per Deep Neural Network	35
3.1 Introduzione alla quantizzazione	35
3.1.1 Motivazioni e sfide	41
3.2 Quantizzazione nelle Deep Neural Network	47
3.2.1 Post-Training Quantization	48
3.2.2 Quantization Aware Training	52
3.2.3 Confronto tra PTQ e QAT	54

3.3 Quantizzazione in MADRL	55
4 Metodologia	56
4.1 Communication System Model	57
4.1.1 Traffic model	58
4.2 Framework MARL adottato	58
4.2.1 Procedure di training e testing	60
4.2.2 Policy DNN	62
4.3 KPI	63
4.3.1 KPI di comunicazione	63
4.3.2 KPI di efficienza	65
4.4 Applicazione della quantizzazione	65
4.4.1 Applicazione della quantizzazione dinamica	66
4.4.2 Applicazione della quantizzazione statica	68
4.4.3 Applicazione della Training Aware Quantization	74
5 Risultati ottenuti	78
5.1 Risultati - Dynamic quantization	78
5.2 Risultati - Static quantization	81
5.2.1 Risultati della divergenza di Kullback-Leibler	81
5.2.2 Risultati - Quantizzazione statica - Media/Mediana	118
5.2.3 Risultati - Quantizzazione statica	121
5.2.4 Risultati - Quantizzazione statica - Custom Config	123
5.3 Risultati - Quantization Aware Training - Default	124
5.4 Risultati - Quantization Aware Training - Custom	127
5.5 Risultati - Confronto fra i diversi tipi di quantizzazione	129
6 Conclusioni	131

Capitolo 1

Reti 6G ed Industrial Internet of Things

In quasi 50 anni, le reti mobili hanno subito un'evoluzione straordinaria, con progressi notevoli in termini di velocità, latenza e capacità di connessione. In questo contesto, il 6G si prefigura come il prossimo upgrade, ridefinendo completamente il concetto di rete mobile. Inizialmente, questo capitolo descriverà l'evoluzione delle reti mobili, partendo dalle prime generazioni analogiche (1G) fino alla quinta generazione (5G). Successivamente, approfondirà i principi fondamentali del 6G, come la sua architettura, le tecnologie che lo renderanno possibile, i suoi casi d'uso e le sfide che dovrà affrontare.

1.1 Storia delle reti mobili

Prima di approfondire il concetto di reti 6G, è opportuno definire il termine *reti mobili* e analizzare come queste si siano evolute nel corso della loro storia. Una rete viene definita mobile se:

- Ha un accesso wireless;
- Se supporta la mobilità degli utenti (o degli endpoint); quindi, se un utente cambia *access point*, quest'ultimo deve rimanere in grado di accedere alla rete.

I principali obiettivi iniziali delle reti mobili erano consentire la mobilità degli utenti e la comunicazione con gli utenti della rete fissa. Inizialmente, il traffico circolante nelle prime reti mobili era principalmente vocale ma, con il passare degli anni, a causa della diffusione di applicazioni come giochi e social network, le reti si sono evolute per gestire prevalentemente un traffico di tipo dati.

La storia delle reti mobili comincia negli anni '80 con quelle di prima generazione. Queste, permisero agli utenti di effettuare chiamate in mobilità grazie all'ausilio dei segnali analogici. Uno dei principali problemi delle reti di prima generazione però, era la mobilità limitata degli utenti, dovuta dalla mancanza di uno standard comune; ogni regione geografica adottava il proprio standard, e tali standard risultavano incompatibili tra di loro. Risultava quindi impossibile l'utilizzo dei dispositivi mobili al di fuori delle regioni supportate.

Negli anni '90 con la seconda generazione (2G) viene introdotto il segnale digitale. Questo offrì la possibilità di trasferire dati, di migliorare la resistenza alle interferenze e di utilizzare nuove tecniche crittografiche, migliorando così anche la sicurezza delle comunicazioni.

Successivamente, nei primi anni 2000, venne introdotta la terza generazione (3G), che consentì per la prima volta l'accesso a internet in mobilità, rendendo possibile l'utilizzo di nuovi servizi multimediali come la messaggistica istantanea.

L'introduzione della quarta generazione (4G) nel 2008 ha trasformato ulteriormente il settore, offrendo connessioni veloci e stabili in banda larga. Questo ha reso possibile l'utilizzo di applicazioni multimediali come i servizi streaming, le videochiamate e il gaming online [1].

Ad oggi, siamo arrivati alla quinta generazione la cui distribuzione risale al 2019. Il 5G permette di utilizzare tecnologie avanzate come la realtà aumentata e virtuale per esperienze immersive nei videogiochi o per applicazioni professionali, come la manutenzione di impianti industriali attraverso visori AR che forniscono istruzioni in tempo reale. Con la quinta generazione è cambiato il soggetto a cui è rivolta la rete, non si parla più di utenti ma di settori di applicazioni (es: automotive, sanità, energia). L'obiettivo del 5G è fornire a tali settori/mercati dei servizi mirati mediante una softwareizzazione delle funzioni di rete ottenuta tramite tecnologie come *Software Defined Networking* (SDN), *Network Function Virtualization* (NFV) e *Cloud Computing*, che realizzano reti verticali virtuali adeguate ai servizi tramite il meccanismo del *network slicing* [2, 3, 4, 5, 6, 7].

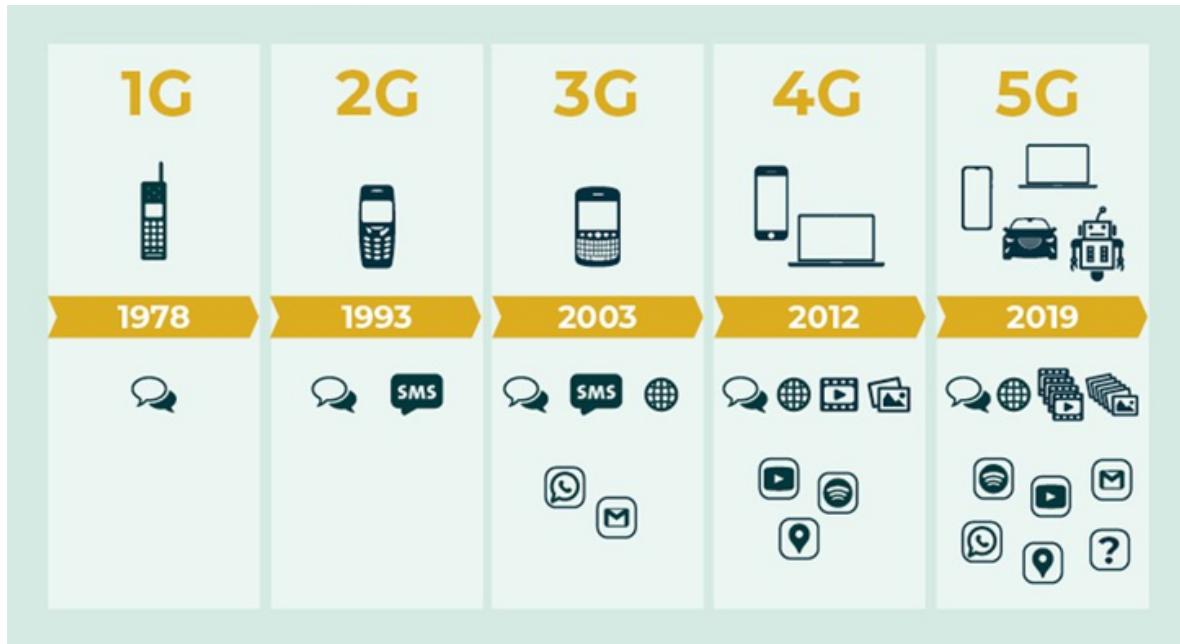


Figura 1.1: Evoluzione delle reti radio mobili [8].

1.2 Introduzione al 6G

Attualmente, il 6G (o IMT-2030) è previsto per il 2030 e sarà progettato sulla base di 5 principi chiave fondamentali [9]:

1. **Integrazione multirete 3D:** La rete 6G sarà progettata per integrare diversi tipi di reti, comprendendo quelle terrestri, spaziali, aeree marittime, subacquee e sotterranee. Questa garantirà una connettività resiliente anche in condizioni estreme, trasformando il 6G in una rete *3D full-space* (copertura totale).
2. **Sicurezza e affidabilità:** Con l'incremento della softwarizzazione delle infrastrutture, la sicurezza sarà un elemento imprescindibile della progettazione delle reti 6G. La protezione sarà integrata fin dalla fase di sviluppo per affrontare le crescenti minacce informatiche e garantire elevati standard di affidabilità.
3. **Integrazione di comunicazioni, rilevamento ed elaborazione:** La rete 6G supererà le tradizionali capacità di comunicazione, includendo funzionalità come il rilevamento, l'*advanced imaging* e il posizionamento preciso supportati da tecnologie come il *cloud* e l'*edge computing*.

4. **Progettazione ecologica, flessibile e leggera:** Per rispondere alle sfide della sostenibilità e dell'efficienza energetica, il 6G utilizzerà tecniche innovative. Tra queste spiccano l'architettura cell-free, la convergenza tra *Radio Access Network* (RAN) e *Core*, e l'adozione di architetture disaccoppiate.
5. **Intelligenza artificiale nativa:** L'intelligenza artificiale sarà integrata nativamente nella rete, svolgendo un ruolo essenziale. Si distingueranno due principali applicazioni:
 - **AI for the Network:** Per supportare la manutenzione e l'ottimizzazione delle reti.
 - **The Network for AI:** Per fornire risorse e infrastrutture adeguate all'intelligenza artificiale.

La sesta generazione punta ad aumentare la velocità di trasmissione dei dati rispetto al 5G, ad ottenere una latenza nell'ordine del microsecondo, a migliorare le prestazioni degli scenari già presenti, ad aprire le porte a scenari del tutto nuovi e introduce un'architettura di rete ricca di novità tecnologiche. Nelle figure a seguire sono riportati gli elenchi delle capacità della rete di sesta generazione, accompagnati dai relativi valori comparati con quelli della rete 5G.

Technology	5G	6G
Applications	Enhanced Mobile Broadband Communications (eMBB), Ultrareliable Low Latency Communications (URLLC), Massive Machine Type Communications (mMTC)	Holographic-Type Communication (HTC), Tactile Internet, Intelligent Transport and Logistics, Intelligent and automated machines, Virtual Reality (VR), Augmented Reality (AR), Extended reality (XR)
Peak data rate	10 Gbps	1 Tbps
Frequency	3–300 GHz	1000 GHz
Latency	10 ms	<1 ms
Mobility support	Up to 500 km/h	Up to 1000 km/h
Spectral efficiency	30 bps/Hz	100 bps/Hz
Reliability	99.9999%	99.99999%

Figura 1.2: Confronto tra 6G e 5G [10].

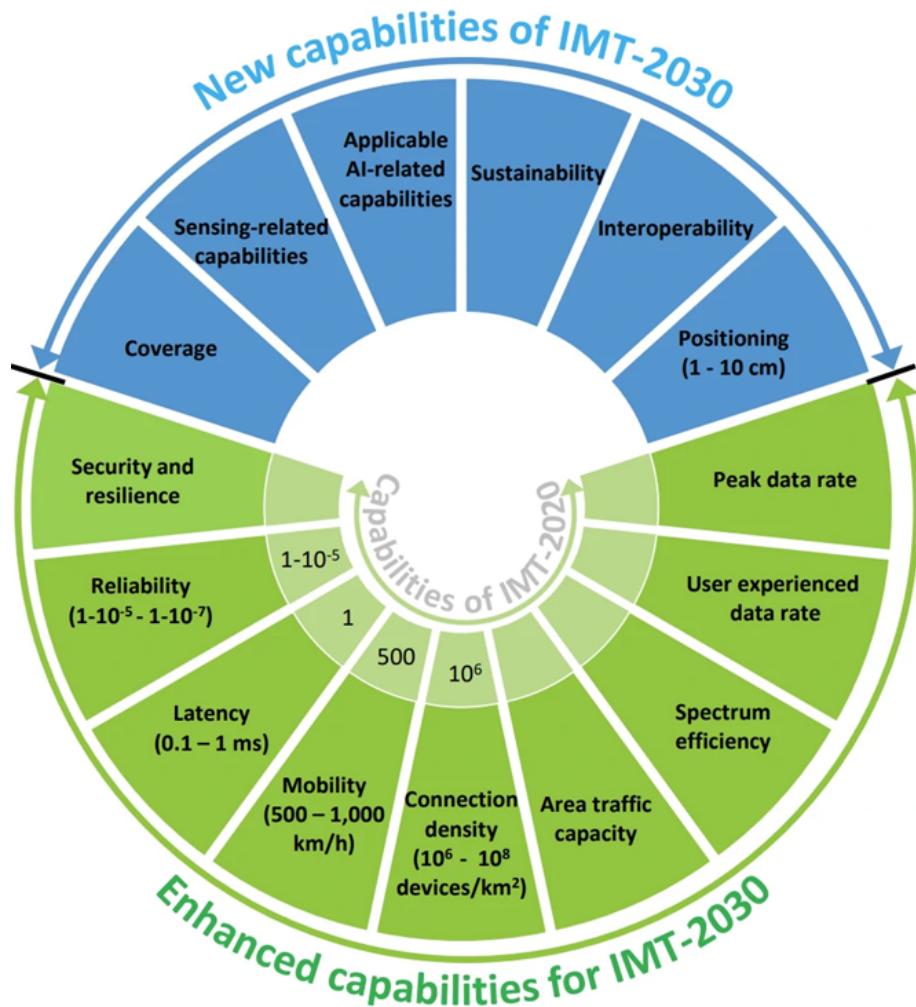


Figura 1.3: Elenco delle capacità delle reti 6G a confronto con quelle del 5G [9].

1.3 Scenari d'uso 6G

Come già accennato nel paragrafo precedente, il 6G, oltre a migliorare le prestazioni degli scenari introdotti dalla generazione precedente, aprirà la strada a scenari del tutto nuovi, che saranno resi possibili da tecnologie avanzate come l'intelligenza artificiale e il *sensing*, tra cui [9]:

- **Comunicazione immersiva:** Il 6G porterà l'*Enhanced Mobile Broadband* (eMBB) del 5G a un nuovo livello, offrendo esperienze video interattive e possibili interazioni avanzate con le interfacce macchina. Ciò richiederà una *Quality of service* (QoS) uniforme, alta affidabilità e bassa latenza.

- **Comunicazione ultra-affidabile ed a bassa latenza:** Con la sesta generazione, l'*Ultra-Reliable and Low-Latency Communication* (URLLC) offerta dal 5G verrà migliorata per rispondere a esigenze più stringenti, supportando anche operazioni sincronizzate *Hard-Real Time*, dove un piccolissimo errore potrebbe avere conseguenze critiche. Ciò richiederà affidabilità estrema e latenza ridottissima.
- **Comunicazione di massa:** Il 6G estenderà il *massive Machine Type Communication* (mMTC) del 5G, connettendo un numero ancora più grande di dispositivi e sensori. Questo sarà fondamentale per applicazioni che spaziano dall'IoT alle *smart city*. La tecnologia dovrà garantire connessioni dense, consumi energetici ridotti, copertura estesa e sicurezza elevata.
- **Connettività obliqua:** Un altro degli obiettivi del 6G è ridurre, grazie all'interoperabilità con altri sistemi, il divario digitale, assicurando una connessione fluida anche in aree rurali o remote attualmente non coperte o scarsamente servite.
- **Intelligenza artificiale e comunicazione:** In questo scenario si assisterà all'integrazione dell'AI con le reti di comunicazione, facilitando applicazioni come la guida autonoma. Questo scenario richiederà affidabilità elevata, latenza minima e funzionalità di calcolo distribuito.
- **Integrazione di sensori e comunicazione:** In questo scenario si utilizza IMT-2030 per offrire un rilevamento multidimensionale ad ampia area che fornisce informazioni spaziali su oggetti non connessi, nonché dispositivi connessi e sui loro movimenti e dintorni. Oltre alle capacità di comunicazione fornite, questo scenario d'uso richiede il supporto di funzionalità di posizionamento ad alta precisione e capacità correlate al sensing, tra cui stima di distanza/velocità/angolo, rilevamento di oggetti e presenza, localizzazione, mappatura e imaging.

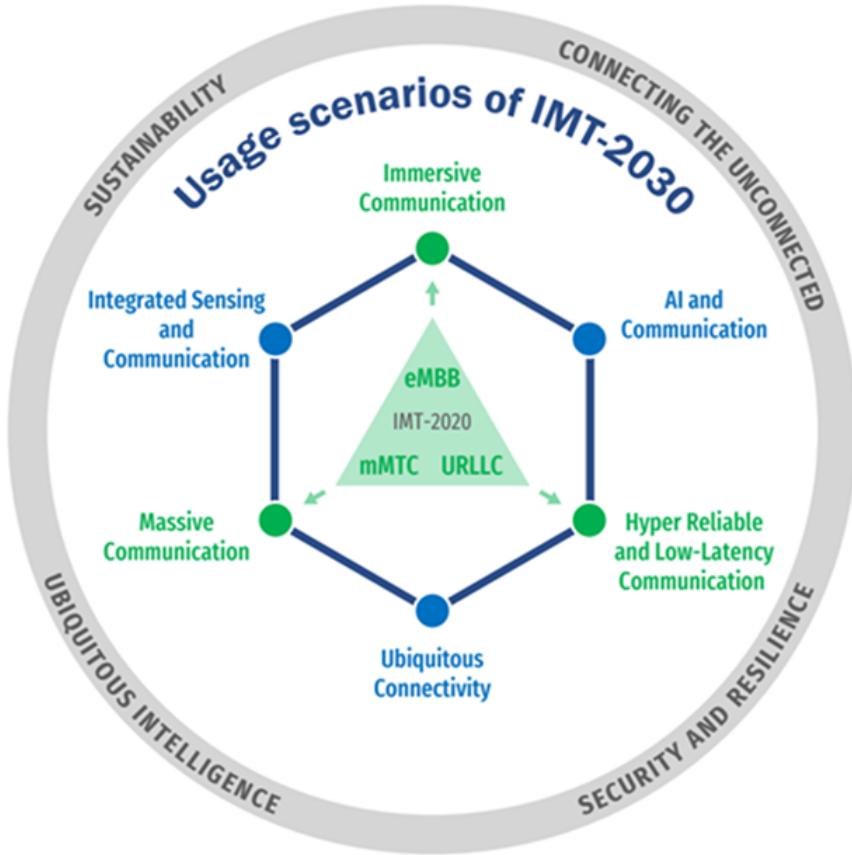


Figura 1.4: Illustrazione degli scenari d'uso di IMT-2030 e confronto con IMT-2020 [9].

1.4 Tecnologie abilitanti

I casi d'uso precedentemente discussi conducono al miglioramento di tecnologie già esistenti e aprono la porta a nuove tecnologie che fino a qualche anno fa potevano sembrare *fantascienza* [9, 11].

Computing ubiquo A partire già dalla quinta generazione, la rete non fornisce solo *pipe per bit*, ma piattaforme per l'*hosting* di applicazioni e servizi. In questo modo si avvicinano i servizi all'utente abbassando la latenza. Il 6G vedrà una diffusione delle risorse di calcolo, con una rete sempre più orientata verso il cloud, consentendo di elaborare informazioni in modo più rapido rispetto al 5G.

Intelligenza ubiqua A causa della continua evoluzione e della rapida diffusione dell'AI e in modo particolare del *machine learning* (ML), si prevede che queste tecnologie saranno presenti in ogni parte del sistema di comunicazione. Le possibili applicazioni dell'AI in questo contesto sono molte, per esempio:

- **Creare dispositivi consapevoli:** I dispositivi connessi potrebbero diventare consapevoli del contesto e svolgere meglio le loro azioni. Per esempio, i dispositivi presenti nelle *smart home* e *smart city* potrebbero adattarsi al tempo o alle abitudini degli utenti.
- **Gestione autonoma delle reti:** Le reti potrebbero essere in grado di monitorarsi, organizzarsi e ottimizzarsi da sole senza l'intervento umano.
- **Miglioramento dell'architettura di rete 6G.**

Digital Twin (DT) Un *digital twin* è un gemello digitale di risorse aziendali come processi, sistemi, prodotti, infrastrutture o dispositivi. Un DT contiene informazioni dettagliate sul corrispondente gemello fisico e modella il suo comportamento dinamico. L'uso dei digital twin nell'Industria si estende lungo l'intero ciclo di vita del gemello fisico, soprattutto nella fase di progettazione e nella fase di operazione e manutenzione. Durante la progettazione, i digital twin (di simulazione) vengono utilizzati per simulare diversi scenari, determinare il design ideale e per verificare eventuali problemi di progettazione ancor prima che la risorsa reale venga realizzata. Nella fase di operazione e manutenzione, i digital twin sono utilizzati per gestire/monitorare impianti e risorse reali di settori industriali.



Figura 1.5: Tony Stark alle prese col modello digitale della sua armatura [12].

Multimedia immersivo e interazioni multisensoriali Si prevede che il futuro della comunicazione multimediale con il 6G offrirà un’esperienza immersiva attraverso interazioni multisensoriali e una profonda integrazione tra il mondo fisico e quello digitale. Questa evoluzione consentirà esperienze di realtà estesa (XR) interattive e in tempo reale. Ad esempio, la telepresenza olografica potrebbe diventare una pratica comune, trasformando il modo in cui viviamo.

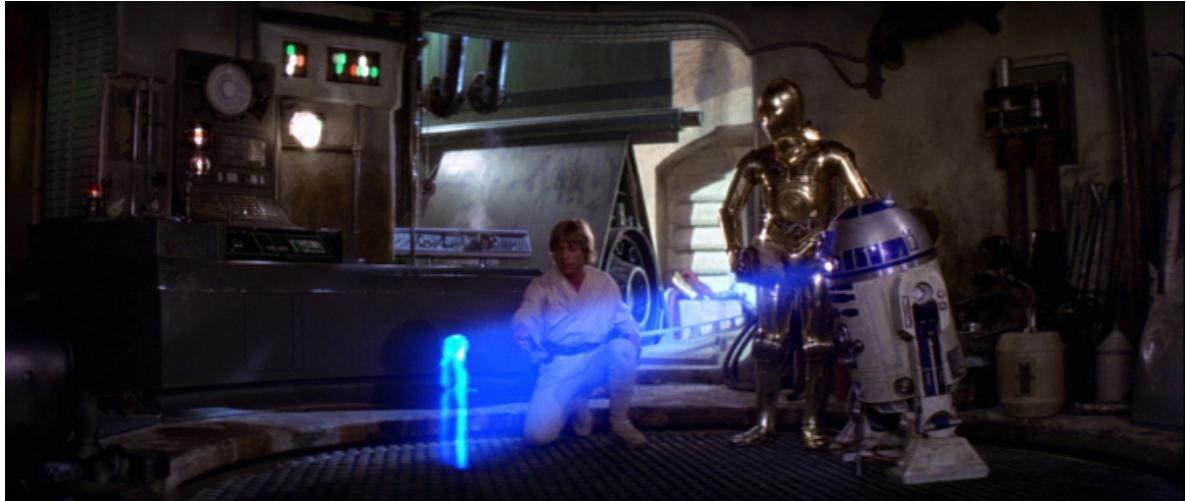


Figura 1.6: Riproduzione di un messaggio olografico in un film della saga di *Star Wars* [13].

Applicazioni industriali intelligenti Il 6G potrebbe rivoluzionare il modo in cui le risorse e l’energia vengono utilizzate nel settore industriale, ottimizzando i processi di produzione, automatizzando la consegna dei prodotti e migliorando l’efficienza. Le industrie avranno bisogno di dispositivi intelligenti con connessioni ultra-affidabili e a bassa latenza, oltre a una capacità di localizzazione precisa, per raccogliere e condividere informazioni in tempo reale, permettendo un controllo intelligente e un feedback continuo. Questo, in combinazione ai già citati digital twin, migliorerà ulteriormente l’efficienza e la gestione delle operazioni industriali. Per esempio, in un magazzino intelligente, robot autonomi e consapevoli dell’ambiente potrebbero muoversi senza problemi tra gli scaffali per raccogliere e consegnare rapidamente i prodotti adattandosi continuamente alle condizioni dell’ambiente.



Figura 1.7: Un robot corriere intelligente [14].

Salute digitale Grazie all'intelligenza artificiale, all'edge computing, alla connettività sempre attiva e ai sensori avanzati, il 6G potrà rendere i servizi sanitari ancora più accessibili e completi, permettendo operazioni come: tele-diagnosi, monitoraggio remoto, assistenza medica a distanza (come le ambulanze connesse), riabilitazione a distanza, studi clinici digitali e molto altro. Inoltre, con l'aumento dei dispositivi indossabili e dei sensori corporei connessi, la salute digitale diventerà sempre più pervasiva, rendendo il monitoraggio del nostro stato di salute un'attività quotidiana. In alcuni casi, questi dispositivi potrebbero addirittura funzionare senza la necessità di batterie, sfruttando altre fonti di energia. Questa tecnologia potrebbe rivoluzionare la salute digitale consentendo un monitoraggio sanitario continuo, a lungo termine e senza interruzioni.

1.5 Architettura della rete 6G

L'architettura di una rete mobile si divide in due sottoreti fondamentali [1]:

- **La Core Network:** Una rete di trasporto che contiene elementi di instradamento delle informazioni. Tale rete si è evoluta in velocità grazie ai collegamenti in fibra ottica.
- **La Radio Access Network:** Una rete che comprende gli endpoints e gli access point. L'accesso a tale rete è migliorato negli anni, mediante lo sviluppo di nuove tecnologie wireless e mobili.

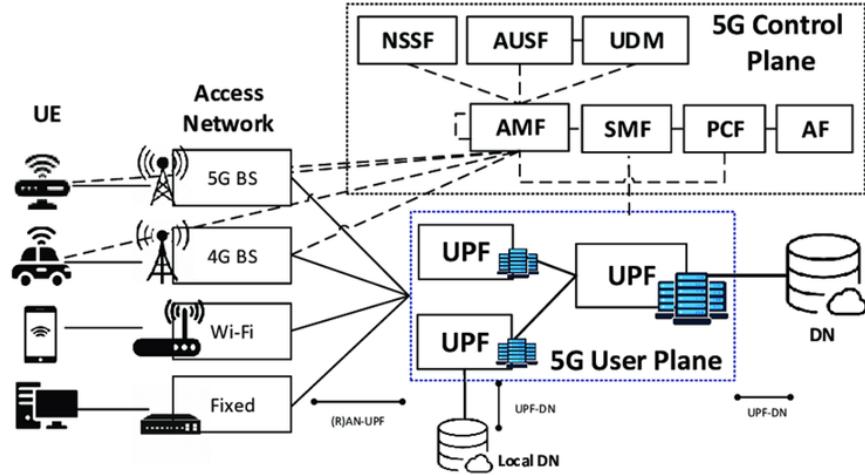


Figura 1.8: Architettura della rete 5G [15].

1.5.1 Core Network

La core 6G sarà completamente decentralizzata rispetto a quella del suo predecessore grazie alla combinazione di nuove tecnologie (AI) e di tecnologie già consolidate dal 5G (es. NFV, SDN), orientate verso i cinque fattori chiave già citati nel paragrafo 1.2 che la rete 6G dovrà possedere. Tra queste tecnologie, descritte in dettaglio in [2, 3, 4, 5, 6, 7], troviamo:

- **Network Function Virtualization (NFV):** Questa tecnologia permette di virtualizzare le funzioni di rete, che in passato erano eseguite su hardware dedicati, trasformandole in software operanti su server standard. Tale approccio consente una maggiore scalabilità e una significativa riduzione dei costi, migliorando l'efficienza complessiva.

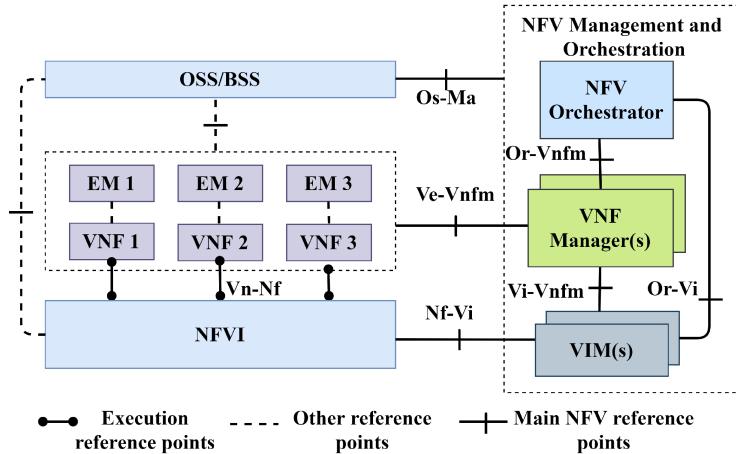


Figura 1.9: NFV architectural framework [5].

- **Software Defined Networking (SDN):** L'SDN separa il piano di controllo dal piano dati, rendendo la gestione della rete centralizzata e programmabile. Questo approccio semplifica la configurazione, rendendo le reti più flessibili e adattabili.

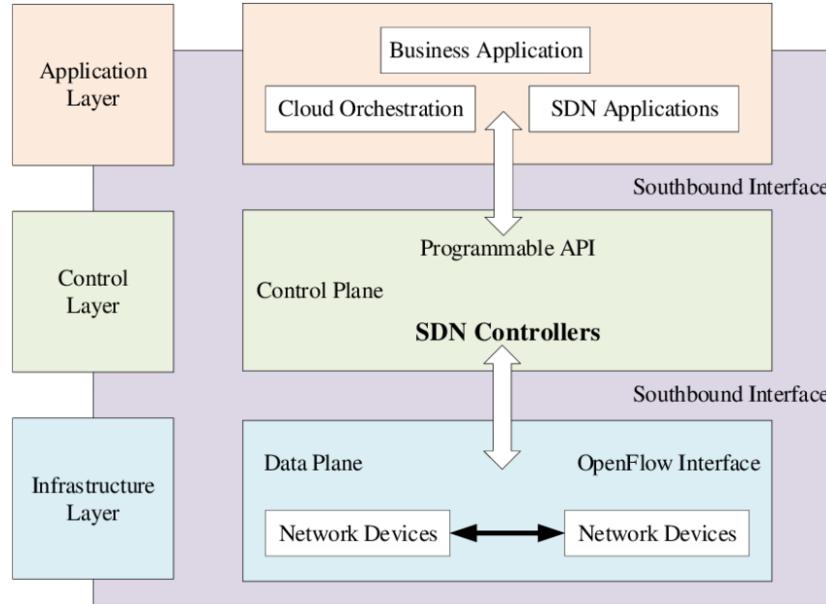


Figura 1.10: SDN basic architecture [6].

- **Architettura basata sui servizi (SBA):** Per affrontare la crescente complessità delle applicazioni, l'SBA suddivide le funzioni di rete in moduli separati che possono essere combinati per supportare diversi servizi. Già adottata nelle reti core 5G, l'SBA sarà estesa alla *Radio Access Network* nel 6G, creando un'architettura *End-to-End* completamente modulare.

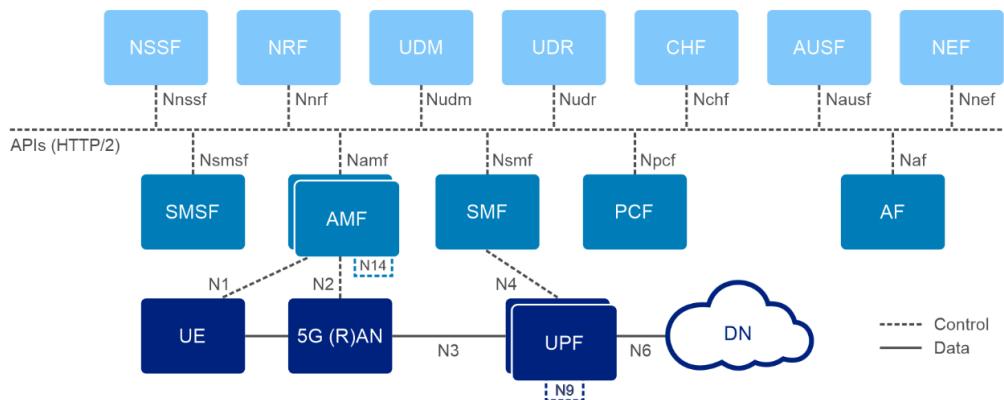


Figura 1.11: SBA architectural [7].

- **Network slicing:** Il Network Slicing permette di creare delle reti logiche indipendenti sulla stessa infrastruttura fisica, ognuna ottimizzata per servizi specifici. Questo migliora l'efficienza nell'utilizzo delle risorse e garantisce una maggiore personalizzazione dei servizi.

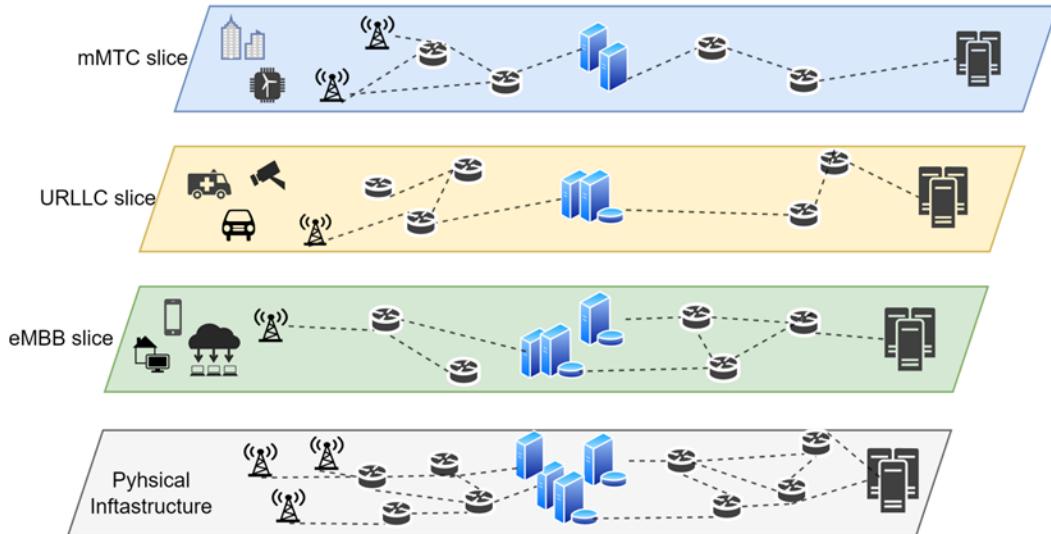


Figura 1.12: Esempio di slicing network [4].

1.5.2 Radio Access Network

La Radio Access Network del 6G sarà caratterizzata dall'introduzione, in diversi settori, di svariate novità, tra cui [2, 16]:

- **Innovazioni dello spettro:** La principale novità è l'introduzione della *banda THz*, con frequenze nel range tra 0,1 THz e 10 THz, e lunghezze d'onda tra 0,03 mm e 3 mm. Tuttavia, la distanza di comunicazione coperta è molto ridotta a causa di problemi ancora irrisolti, tra cui:
 - L'assorbimento del segnale da parte di molecole atmosferiche (es: Ossigeno, vapore acqueo) riduce la portata del segnale;
 - Elevate perdite dovute alla natura intrinseca del segnale, quali scattering, riflessione, dispersione e Shadowing;
 - Limitazione dei dispositivi hardware attuali che non riescono a gestire frequenze così elevate;

Un'altra idea è quella utilizzare le *Comunicazioni Ottiche Wireless* (OWC) che sfruttano le frequenze nella banda ottica, come l'Infrarosso (IR), la luce visibile e l'Ultravioletto (UV), offrendo alla comunicazione:

- Latenza ultra-bassa;
- Sicurezza intrinseca al livello fisico;
- Nessuna interferenza elettromagnetica;
- Spettro non licenziato libero;
- Costi bassi;
- Semplicità di distribuzione.

Tuttavia, il loro utilizzo è ancora limitato dal lato hardware, a causa della larghezza di banda elettrica dei dispositivi optoelettronici.

- **Utilizzo di nuove forme d'onda:** Nel 6G verranno introdotte nuove forme d'onda come l'*Orthogonal Time Frequency Space* (OTFS), la *Spectrally Efficient Frequency Domain Multiplexing* (SEFDM) o l'*Overlapped X Domain Multiplexing* (OVXDM).
- **Introduzione di nuove tipologie di modulazioni:** Per quanto riguarda le modulazioni, in futuro ne verranno considerate di nuove come *Quadrature Amplitude Modulation* (QAM) selezionate, QAM irregolari, interpolazione di costellazioni, modulazione multidimensionale e *Index Modulation*.
- **Nuove tecniche di codifica:** Gli *Error-Correcting Codes* (ECC) sono i codici tipicamente utilizzati nelle reti 4G e 5G. Questi codici introducono casualità nella codifica, permettendo di avvicinarsi al limite di Shannon, rendendoli adatti alle esigenze del 6G in termini di velocità dei dati. Tuttavia, poiché il 6G richiede anche latenza ultra-bassa e ultra-affidabilità, dovrebbero essere adottati codici a lunghezza ridotta, causando una degradazione delle prestazioni degli algoritmi di decodifica ECC. Per superare questo problema, sono considerate tecniche di decodifica come la decodifica a massima verosimiglianza o la *Guessing Random Additive Noise Decoding* (GRAND) che raggiunge la capacità.
- **Nuove tecniche di accesso radio:** Con il termine accesso multiplo al canale si intendono tutte quelle tecniche che consentono a più utenti di accedere e condividere contemporaneamente le risorse del canale trasmissivo. Finora, questo è

stato gestito da tecnologie *Orthogonal Multiple Access*(OMA), come l'*Orthogonal Frequency Division Multiple Access* (OFDMA). Poiché la densità di connessione nelle reti 6G sarà maggiore rispetto alle 5G, dovrebbero essere sfruttate nuove tecnologie per migliorare la capacità e le prestazioni della rete. A tal fine, stanno emergendo le tecnologie *Non-Orthogonal Multiple Access* (NOMA). Il NOMA mira a utilizzare le stesse risorse radio, in tempo, frequenza o codice, per più utenti. Questo dovrebbe aumentare la e migliorare il *throughput* degli utenti ai margini della cella.

- **Utilizzo di Ultra-massive MIMO:** Nei sistemi 6G, il numero di antenne utilizzate sarà dell'ordine di centinaia o migliaia, portando al concetto di ultra-massive MIMO. Con l'*Ultra-massive Multiple Input Multiple Output*, o Ultra-massive MIMO, la scala delle antenne verrà aumentata, ottenendo una migliore risoluzione spaziale, maggiore efficienza spettrale ed energetica, insieme a una copertura di rete più ampia. Tuttavia, l'aumento della scala delle antenne implica che l'approssimazione della fronte d'onda piana non sarà più valida e dovranno essere prese in considerazione le onde sferiche. L'ultra-massive MIMO sfrutterà frequenze più elevate rispetto al massive MIMO e nuove tecnologie come le *Reconfigurable Intelligent Surface* (RIS). Inoltre, poiché l'ultra-massive MIMO è destinato a regolare i fasci in tre dimensioni potrà fornire una copertura non terrestre; quindi, verrà combinata con reti integrate spazio-aria-terra-mare. Infine, si prevede che la tecnologia ultra-massive MIMO venga combinata con l'AI per fornire intelligenza nell'estimazione del canale, nella gestione dei fasci e nel rilevamento degli utenti.
- **Introduzione delle RIS:** Le RIS sono superfici intelligenti riconfigurabili composte da metamateriali 2D programmabili a sub-lunghezza d'onda che possono manipolare intelligentemente il segnale incidente per ottenere il segnale desiderato. Tuttavia, ci sono molte limitazioni da superare, come le capacità hardware, l'architettura delle reti wireless e gli algoritmi di baseband. Inoltre, il consumo energetico di questi dispositivi ad alte frequenze dovrebbe essere ulteriormente analizzato.
- **Nuova tecnica di duplexing:** Verrà esplorata l'*In-Band Full-Duplex* (IBFD) che consente la trasmissione e la ricezione nella stessa banda di frequenza nello stesso momento, che può teoricamente raddoppiare l'efficienza spettrale, espandere la capacità di trasmissione wireless e consentire un accesso più flessibile ed efficiente alla rete. Tuttavia, per rendere l'IBFD efficiente, devono essere introdotte tecniche di cancellazione dell'auto-interferenza (SIC) in quanto con la

trasmissione e la ricezione svolte in maniera simultanea il segnale trasmesso da un dispositivo interferisce con il segnale che lo stesso dispositivo sta cercando di ricevere.

- **Momento Angolare Orbitale (OAM):** Per migliorare ulteriormente l'efficienza spettrale, devono essere esplorate nuove dimensioni fisiche. Le potenziali innovazioni potrebbero includere il Momento Angolare Orbitale, RIS e la radio olografica. Con l'OAM, le onde elettromagnetiche hanno una fase di momento angolare invece di una fase di onda piana. Le antenne genereranno modalità ortogonali, ciascuna con un diverso modo di momento angolare orbitale per trasportare informazioni diverse. Tuttavia, l'utilizzo delle onde elettromagnetiche OAM presenta diverse problematiche che devono essere affrontate:
 - **Divergenza dei fasci e mancato allineamento:** Le onde OAM hanno una struttura a spirale che tende a divergere più rapidamente rispetto ai fasci convenzionali. Inoltre, il corretto funzionamento delle onde OAM richiede un allineamento preciso tra trasmettitore e ricevitore. Anche piccoli errori di allineamento possono portare a una perdita di segnale significativa, riducendo la distanza utile di trasmissione.
 - **Riflessioni e rifrazioni:** Quando le onde OAM incontrano superfici che causano riflessioni o rifrazioni, la loro struttura a spirale può essere alterata. Questo può distruggere l'ortogonalità delle modalità OAM, che è fondamentale per separare i segnali e mantenere l'integrità dei dati.

1.6 Intelligenza artificiale nella rete 6G

Nel contesto delle reti 6G, l'AI verrà applicata sia nella RAN che nella Core Network; In particolare l'AI gestirà sfide tra cui [16]:

- **Stima del canale:** La stima del canale è il processo con cui si misura come il segnale trasmesso viene alterato durante la sua propagazione (ad esempio, a causa di rumore, ostacoli o riflessioni). Tale stima permette al ricevitore di compensare le alterazioni e recuperare correttamente il segnale originale. Le tecnologie chiave del 6G aumenteranno sia in termini numerici che di latenza la complessità di calcolo di questa stima che non potrà più essere effettuata con i tradizionali approcci matematici. In questa casistica l'AI saranno utilizzate per ottenere una stima del canale veloce e accurata.

- **Riconoscimento della modulazione:** Determinare la modulazione utilizzata su un segnale è fondamentale per recuperare correttamente i dati trasmessi. La complessità di questo riconoscimento con il 6G aumenterà a causa delle maggiori interferenze dovute all'elevata densità di dispositivi e alla bassa copertura intrinseca del segnale. In questo contesto, l'AI garantirà un riconoscimento della modulazione rapido e preciso.
- **Classificazione del traffico:** Classificare il traffico è necessario per garantire la giusta QoS a ogni flusso ed a permettere una corretta gestione delle risorse. Tradizionalmente, viene realizzato mappando il traffico al numero di porta o tramite metodi basati sul *payload* dei pacchetti. Tuttavia, entrambi gli approcci presentano dei limiti; il primo è limitato dalla dinamicità delle porte, mentre il secondo risulta inefficace in presenza di crittografia in quanto rende inaccessibile il contenuto dei pacchetti.
- **Predizione del traffico:** Per gestire al meglio le risorse e migliorare la QoS, l'Intelligenza Artificiale verrà utilizzata per analizzare i dati sul traffico conosciuti e prevedere l'andamento futuro della rete.
- **Routing intelligente:** Il compito delle tecniche di *routing* è analizzare un'enorme quantità di informazioni per trovare percorsi ottimali e gestire dinamicamente i cambiamenti nella rete. A causa della crescita esponenziale dei dispositivi connessi alla rete, le tecniche di routing tradizionali stanno diventando sempre più onerose dal punto di vista computazionale risultando poco scalabili. Quindi per avere un routing più efficiente, veloce e scalabile rispetto ai metodi tradizionali si utilizzeranno soluzioni basate su AI, che saranno in grado di apprendere automaticamente dai dati della rete e di adattarsi rapidamente alle sue variazioni.
- **Gestione delle risorse radio e della mobilità;**
- **Ottimizzazione energetica;**
- **Sicurezza e privacy.**

Capitolo 2

Multi-agent deep reinforcement learning

Il *reinforcement learning* (RL) rappresenta un ramo del ML in cui un agente apprende come compiere decisioni ottimali attraverso interazioni con l’ambiente, ricevendo ricompense o penalità in base alle proprie azioni. Basandosi sui principi fondamentali del RL, il *multi-agent reinforcement learning* (MARL) si concentra su scenari in cui più agenti operano nello stesso ambiente, interagendo tra loro per raggiungere obiettivi comuni o individuali. Questo approccio introduce sfide complesse, tra cui la necessità di coordinare le azioni tra agenti, gestire le dipendenze reciproche e garantire la stabilità durante il processo di apprendimento. Un’ulteriore specializzazione del MARL è rappresentata dal *multi-agent deep reinforcement learning* (MADRL), che integra l’uso delle *Deep Neural Network* (DNN) per migliorare la capacità di apprendimento degli agenti. In questo capitolo verranno inizialmente presentate le basi del RL e le sue principali categorie di algoritmi. Successivamente, l’attenzione si focalizzerà sull’approfondimento degli approcci MARL e MADRL.

2.1 Reinforcement Learning

Il Reinforcement Learning è caratterizzato da uno o più agenti che apprendono in autonomia il comportamento ottimale da adottare attraverso l’interazione con l’ambiente. Per capire meglio, dato uno stato del sistema s_t , l’agente eseguirà un’azione x_t , dove t è riferito ad un determinato istante temporale. Eseguendo l’azione a_t l’agente andrà ad alterare lo stato del sistema da s_t a s_{t+1} . In base al nuovo stato raggiunto dal sistema l’agente riceverà dall’ambiente una *reward* (o ricompensa) r_t , che può essere negativa o positiva. L’obiettivo dell’agente è massimizzare le rewards cumulative ottenute dal-

l’ambiente. Un esempio di apprendimento basato su rinforzo potrebbe essere quello di un cane (agente) che impara riportare indietro un bastone lanciato dal petmate (ambiente). Se il cane dopo aver lanciato il bastone lo afferra e lo riporta indietro, gli si dà un biscotto (reward). Così facendo, con il tempo il cane capirà che riportando indietro il bastone guadagnerà le sue ricompense [17].

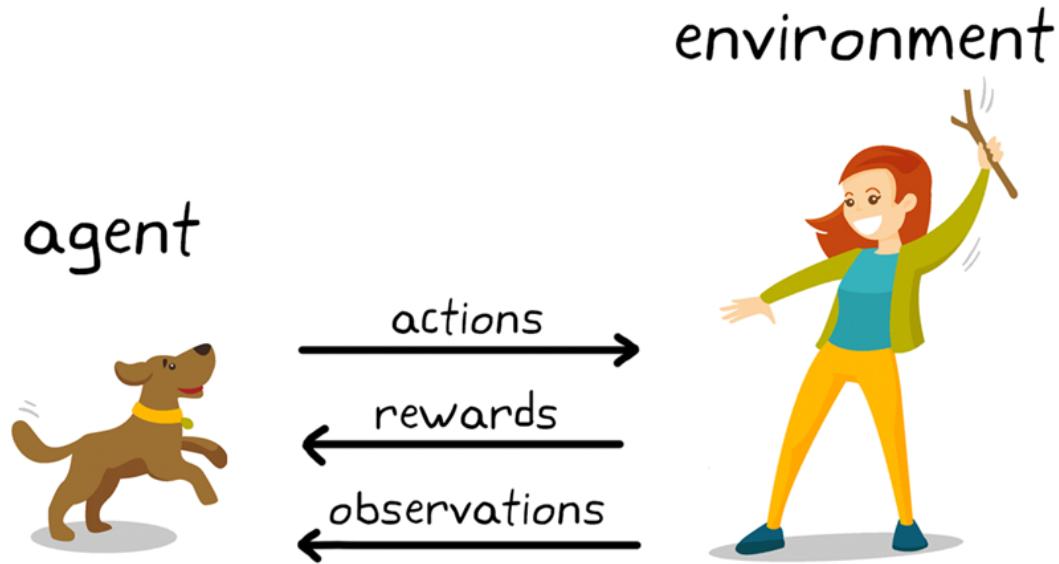


Figura 2.1: Esempio reale di Reinforcement Learning [17].

Nel Reinforcement Learning, l’agente sceglie l’azione a_t da eseguire mediante una *policy*, cioè, una funzione $\pi : S \rightarrow A$, dove S rappresenta l’insieme degli stati possibili e A rappresenta l’insieme delle azioni disponibili. La policy mappa ogni stato ad un’azione (policy deterministica, $a = \pi(s)$, oppure ad una distribuzione di probabilità sulle azioni (policy probabilistica, $a \sim \pi(a | s)$).

L’obiettivo è determinare una *policy ottimale* che non si limiti a massimizzare la reward immediata, ma consideri anche quella complessiva accumulata nel tempo. Poiché le azioni intraprese dall’agente possono produrre esiti diversi a causa della natura stocastica dell’ambiente, non è sempre possibile massimizzare la reward in ogni singolo episodio. Di conseguenza, l’agente mira a massimizzare l’*expected return*, cioè il valore medio del ritorno su più episodi, calcolato come:

$$\mathbb{E}_\pi[r_0 + r_1 + \dots + r_{T-1}]. \quad (2.1)$$

Nel Reinforcement Learning, l’interazione tra l’agente e l’ambiente può essere formalizzata matematicamente come un *Markov decision process* (MDP), una struttura che descrive gli stati, le azioni, le transizioni e le ricompense. Per gli MDP che terminano (cioè in cui l’episodio finisce dopo un certo numero di passi T), questa somma di ricompense è sempre finita mentre per gli MDP non terminanti (dove non c’è un limite nel numero di passi), la somma delle ricompense potrebbe essere infinita. In questi casi, per evitare che i ritorni diventino infiniti e per garantire che la valutazione delle policy sia significativa, si usa un *discount factor* γ , con $\gamma \in [0, 1]$. Questo fattore riduce l’importanza delle ricompense future rispetto a quelle immediate. Quando γ è vicino a 1, l’agente dà quasi la stessa importanza a tutte le ricompense future, mentre se γ è più piccolo (ad esempio 0.9 o 0.5), l’agente dà più peso alle reward immediate rispetto a quelle future.

$$\mathbb{E}_\pi [r_0 + \gamma r_1 + \gamma^2 r_2 + \dots] = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \right] = \mathbb{E}_\pi [G_t]. \quad (2.2)$$

Il valore calcolato prende il nome di *discounted return* e massimizzandolo si può trovare la policy ottimale:

$$\pi_* = \arg \max_{\pi} \mathbb{E}_\pi [G_t]. \quad (2.3)$$

Uno strumento fondamentale per trovare la policy ottimale è la *Value function*, poiché permette all’agente di stimare l’expected value a partire da uno stato s e dalla policy π . La value function per uno stato s , è data da:

$$V_\pi(s) = \mathbb{E}_\pi[G_t | s_t], \quad (2.4)$$

Come già accennato, l’azione scelta dall’agente determina una transizione dello stato dell’ambiente. Questa transizione è descritta da una *transiction function* T (o funzione di transizione), che rappresenta la probabilità di passare dallo stato s_t allo stato s_{t+1} a seguito dell’azione a_t , espressa come $T(s_{t+1} | s_t, a_t)$.

Sulla base dello stato raggiunto, l’ambiente assegna una reward tramite una *reward function* (o funzione di ricompensa) $R(s_t, a_t, s_{t+1})$, che restituisce un valore numerico che misura quanto l’agente ha agito correttamente o meno in quella situazione. Infine, in base alla reward ricevuta e al nuovo stato raggiunto l’agente aggiorna la propria policy in modo da migliorare le sue decisioni future. Per fare ciò si utilizzano diversi algoritmi che verranno discussi in seguito [18].

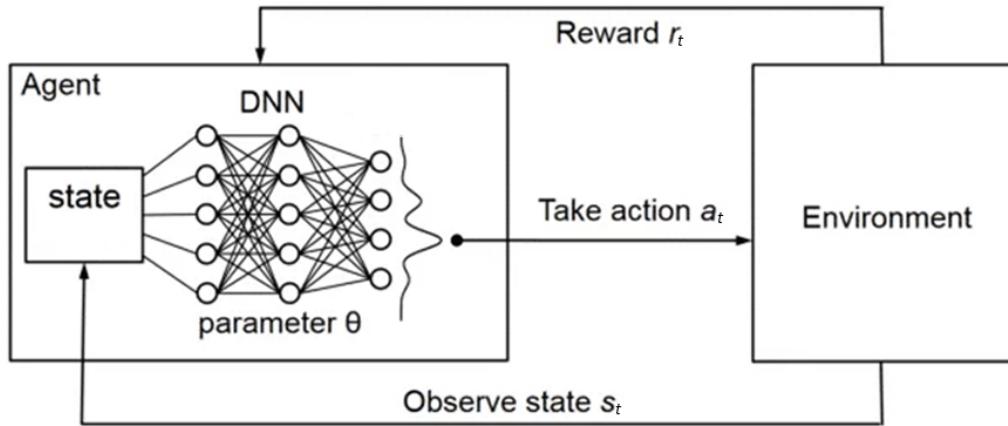


Figura 2.2: Schema di funzionamento del Reinforcement Learning [19].

2.1.1 Algoritmi di Reinforcement Learning

Gli algoritmi di Reinforcement Learning possono essere classificati in *model based* e *model free*; I primi sfruttano dei modelli per simulare l'ambiente, mentre i secondi apprendono senza l'uso di modelli, basandosi solo sulle esperienze reali [20].

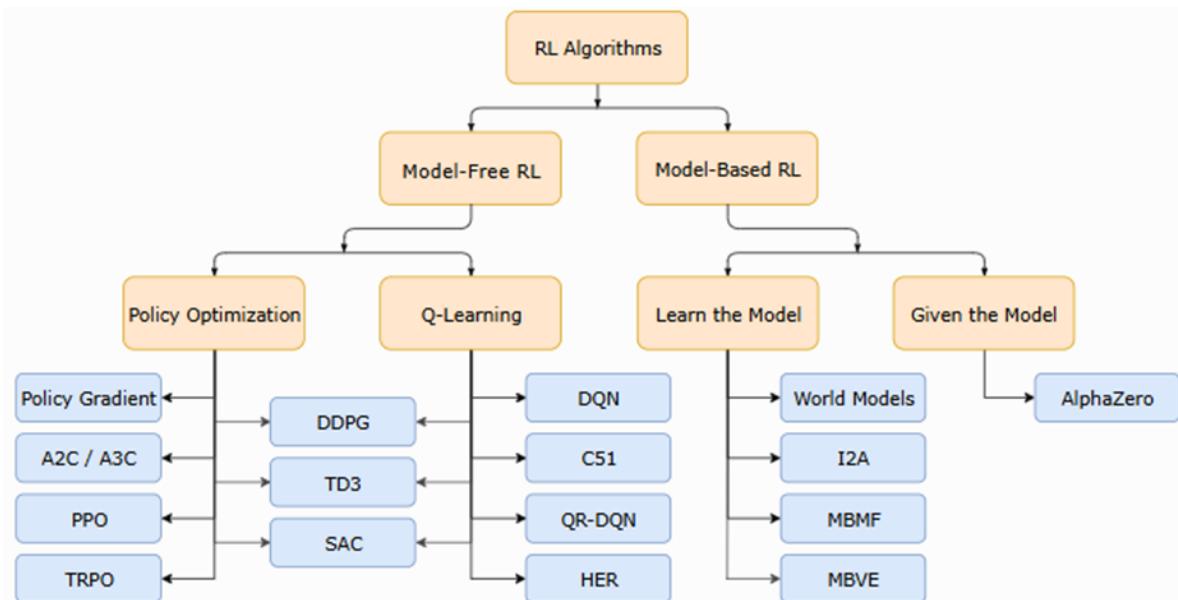


Figura 2.3: Classificazione degli algoritmi di Reinforcement Learning [20].

Model free

Come illustrato in figura 11, gli algoritmi model free si dividono in due sottocategorie: *Policy Optimization* e *Q-Learning*.

Policy Optimization Nella Policy Optimization l’obiettivo è quello di apprendere direttamente la policy ottimale. Questa viene quasi sempre eseguita *on-policy*, il che significa che ogni aggiornamento utilizza solo i dati raccolti mentre l’agente agisce secondo l’ultima versione della policy. Tra gli algoritmi più utilizzati troviamo [18, 21, 22]:

- **Policy Gradient (PG)**: Il PG punta a creare una policy che massimizza l’expected value a partire da un certo stato o azione, tenendo conto della policy seguita. In questo contesto la policy viene rappresentata come una funzione parametrizzata i cui parametri vengono aggiornati per migliorare la performance dell’agente. Questi aggiornamenti vengono effettuati utilizzando il gradiente della funzione obiettivo (che rappresenta l’expected value) rispetto ai parametri della policy, calcolato tramite il *Policy Gradient Theorem*.
- **Proximal Policy Optimization (PPO)**: L’obiettivo principale della PPO è migliorare la policy corrente limitando il cambiamento tra la vecchia policy e la nuova policy, in modo che non ci siano cambiamenti troppo drastici che potrebbero destabilizzare l’apprendimento. Per fare ciò l’algoritmo stima tramite una funzione quanto la nuova policy sia migliore della vecchia. Questa funzione prende il nome di *funzione surrogata* e rappresenta il rapporto tra le probabilità assegnate dalla nuova policy e quelle assegnate dalla vecchia per un’azione a_t in uno stato s_t .

$$r_t(\theta) = \frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} \quad (2.5)$$

Per valori di $r_t(\theta) > 1$, la nuova policy assegna una probabilità maggiore all’azione rispetto alla vecchia policy, mentre valori di $r_t(\theta) < 1$ indicano una probabilità minore. Il compito della funzione surrogate è massimizzare questo rapporto, pensandolo con l’*advantage* \hat{A}_t , che rappresenta una stima di quanto l’azione a_t sia migliore rispetto alla media delle azioni disponibili nello stato s_t . Questo permette di aggiornare la policy in modo che privilegi azioni più vantaggiose.

Per affrontare il problema degli aggiornamenti troppo grandi, PPO introduce un meccanismo di *clipping*. Questo meccanismo limita il rapporto $r_t(\theta)$ entro

un intervallo predefinito, controllato da un parametro ϵ . Valori più piccoli di ϵ rendono gli aggiornamenti più conservativi, mentre valori più grandi permettono cambiamenti più significativi.

Combinando i 3 pezzi otteniamo la funzione finale, nota come *funzione obiettivo con clipping*, definita come:

$$L^{\text{clip}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right]. \quad (2.6)$$

Questa prende il minimo tra le due opzioni disponibili, ovvero, il valore puro $r_t(\theta) \hat{A}_t$ e il valore di $r_t(\theta)$ clippato nell'intervallo $[1 - \epsilon, 1 + \epsilon]$ per \hat{A}_t .

- **Actor-Critic:** Questo algoritmo è composto da due componenti principali, un *attore*, che apprende la policy, e un *critico*, che valuta l'efficacia della policy utilizzando una Value Function. L'attore sceglie un'azione basandosi sulla policy corrente; Successivamente, il critico valuta quanto sia buona l'azione scelta dall'attore, calcolando l'expected value. Infine, l'attore utilizza il riscontro per aggiornare la sua policy. In sostanza, c'è una sorta di collaborazione tra attore e critico, il critico aiuta l'attore a migliorare la propria policy fornendo un riscontro sulla qualità dell'azione scelta, mentre l'attore esplora diverse policy per permettere al critico di migliorare la propria Value function in modo da ritornare riscontri più affidabili.

Q-learning I metodi di questa famiglia apprendono una *Q-function* $Q^*(s, a)$, ovvero una funzione che stima il valore atteso di una determinata azione a in uno stato s [18, 23].

$$Q^*(s, a) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a \right], \quad (2.7)$$

dove:

- $Q^*(s, a)$ è il valore ottimale della coppia stato-azione;
- r_t è la reward ottenuta nel tempo t ;
- γ è il discount factor.

Questa viene quasi sempre eseguita *off-policy*, il che significa che ogni aggiornamento può utilizzare dati raccolti in qualsiasi momento durante l'allenamento. Possiamo riassumere brevemente il funzionamento di tale famiglia di algoritmi in quattro punti:

1. A ogni iterazione, l'agente deve scegliere un'azione da eseguire in base allo stato in cui si trova. La scelta dell'azione dipende dalla policy adottata. Tra le policy più adottate in questo scenario abbiamo:
 - **Greedy:** L'agente sceglie l'azione che massimizza la Q-function.
 - **Epsilon Greedy:** Definito ϵ , l'agente sceglie l'azione che massimizza la Q-function con probabilità $1 - \epsilon$, oppure sceglie con probabilità ϵ un'azione casuale.
2. Una volta eseguita l'azione, l'agente osserva la reward immediata e il nuovo stato raggiunto e utilizza la Q-function per calcolare $\max_{a'} Q(s', a')$. Questo valore rappresenta la stima sulla ricompensa massima attesa dall'agente partendo dal nuovo stato s' e scegliendo la migliore azione a' .

$$\max_{a'} Q(s', a') \quad (2.8)$$

3. Calcolato $\max_{a'} Q(s', a')$, l'agente aggiorna la Q-function utilizzando l'equazione di Bellman:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right) \quad (2.9)$$

Dove:

- α è il *learning rate*, che determina quanto velocemente l'agente modifica il valore della Q-function. Un valore di α maggiore implica aggiornamenti più rapidi, ma meno precisi.
 - r è la reward immediata;
 - γ è il discount factor.
4. Una volta che l'agente ha appreso la Q-function ottimale, la policy ottimale può essere ottenuta scegliendo, per ogni stato, l'azione che massimizza la Q-value:

$$\pi^*(s) = \arg \max_a Q(s, a) \quad (2.10)$$

Model Based

Gli algoritmi della famiglia Model based si dividono in due categorie principali: *Learn the Model* e *Given the Model* [24].

Given the Model In questi algoritmi, si assume che il modello dell’ambiente sia già noto o fornito e l’obiettivo è determinare una policy ottimale utilizzando il modello noto e l’ausilio di tecniche di pianificazione come *Value Iteration* o *Policy Iteration*.

Learn the Model In questi algoritmi, si apprende il modello dell’ambiente attraverso l’interazione. L’obiettivo finale è quello di costruire un modello accurato e usarlo per apprendere una policy ottimale. In sostanza, l’agente esplora l’ambiente per stimare:

- La funzione di transizione $T(s_{t+1} | s_t, a_t)$, costruendo una tabella o una rete neurale.
- La reward function $R(s, a)$, calcolando una stima media delle ricompense ricevute.

Una volta appreso il modello, utilizza tecniche di pianificazione come Value Iteration o Policy Iteration per calcolare una policy ottimale.

2.2 Algoritmi MARL

Il Reinforcement Learning può essere categorizzato in base al numeri di agenti che operano sul sistema (agent); Queste categorie sono [25]:

- **Single Agent:** Si tratta di un singolo agente che opera sul sistema che cerca di massimizzare la propria ricompensa;

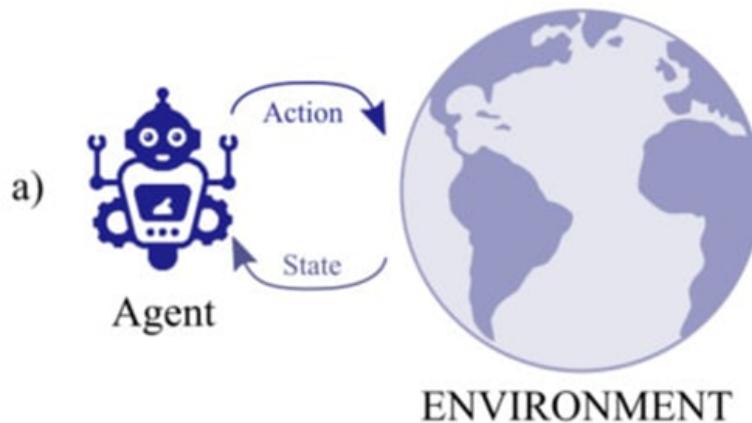


Figura 2.4: Ambiente single agent [25].

- **Multi-agent:** Si tratta di più agenti che operano simultaneamente nello stesso ambiente, e ciascuno cerca di massimizzare la propria ricompensa. Gli agenti possono collaborare, competere o fare entrambe le cose.

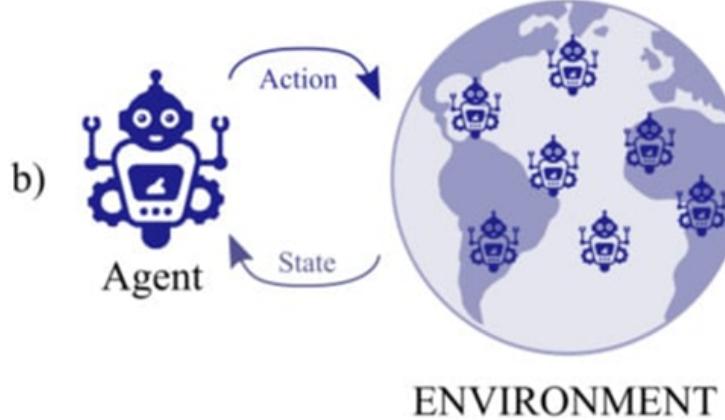


Figura 2.5: Ambiente multi-agent [25].

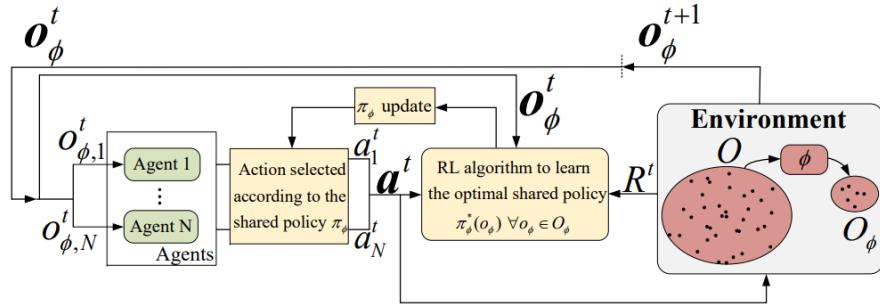


Figura 2.6: Esempio MARL [26].

I concetti introdotti nel paragrafo precedente valgono per il single agent RL ma possono essere estesi nel campo del MARL [26]. I sistemi MARL sono generalmente modellati come *Multi-agent Partially Observable MDP* (MPOMDP), definiti dalla seguente tupla $(S, N, A_i, T, O_i, \pi_i, R, \gamma)$, dove:

- S è lo spazio globale degli stati dell'ambiente. Tuttavia, in un MPOMDP, ogni agente ha accesso solo a una vista parziale di questo stato globale, rappresentato come $o_i^t \in O_i$, che potrebbe non contenere tutte le informazioni sullo stato completo.
- N è l'insieme di tutti gli agenti.

- A_i è lo spazio delle azioni per l'agente i , con $i \in N$. Ad ogni passo temporale t , ciascun agente esegue un'azione $a_i^t \in A_i$.
- T rappresenta la funzione di transizione $T(s_{t+1}|s_t, \mathbf{a}_t)$. In un sistema multi-agente, la probabilità di transizione dallo stato s_t allo stato s_{t+1} dipende dal vettore di azioni congiunte \mathbf{a}_t , che rappresenta l'insieme delle azioni compiute da tutti gli agenti nel sistema:

$$\mathbf{a}_t = (a_1^t, a_2^t, \dots, a_N^t)$$

- O_i è lo spazio delle osservazioni per ciascun agente i . Ad ogni passo temporale t , l'ambiente fornisce ad ogni agente una parte dell'osservazione dello stato $s_t \in S$, definita come $o_i^t \in O_i$.
- $\pi_i : O_i \rightarrow \Delta(A_i)$ è la policy dell'agente i . Il vettore delle policy per ogni agente è definito come $\boldsymbol{\pi} = [\pi_1, \dots, \pi_N]$.
- R rappresenta la reward function $R_{t+1}(s_t, s_{t+1}, \mathbf{a}_t)$ ovvero, la reward ricevuta dopo la transizione dallo stato s_t allo stato s_{t+1} dato il vettore delle azioni congiunte \mathbf{a}_t . Le rewards nei sistemi MARL possono essere sia globali che locali: nel caso di reward globale, la stessa ricompensa viene ricevuta da ciascun agente senza distinguere il contributo individuale, mentre nel caso di reward locale, ogni agente riceve una reward diversa in base al proprio comportamento.
- γ è il discount factor.

Dato un vettore di policy $\boldsymbol{\pi}$, il *discount return* in un sistema MARL è espresso come $\mathbb{E}_{\boldsymbol{\pi}}[G_t]$. La struttura della formula rimane sostanzialmente la stessa di quella indicata nella formula 2.2, ma in questo caso la reward dipende dalle azioni congiunte di tutti gli agenti nel sistema, rappresentate da \mathbf{a}_t . Quindi, in questo caso, l'obiettivo diventa trovare il vettore delle policy ottimale $\boldsymbol{\pi}_*$ tale che:

$$\boldsymbol{\pi}_* = \arg \max_{\forall \boldsymbol{\pi}} \{\mathbb{E}_{\boldsymbol{\pi}}[G_t]\} \quad \forall t \in \{0, 1, \dots, t_{\text{end}}\}. \quad (2.11)$$

L'utilizzo di più agenti introduce nel sistema delle complessità, come [18, 27]:

- **Non stazionarietà:** In caso di Single Agent RL, si assume che l'ambiente sia stazionario, cioè un ambiente con transizioni e ricompense fisse. Mentre, in multi Agent RL questa assunzione non può essere fatta in quanto le azioni di un agente influenzano l'ambiente e, di conseguenza, le osservazioni e i premi ricevuti dagli altri agenti. Questo può essere considerato un problema in quanto gli agenti

potrebbero entrare in un loop di adattamento reciproco, senza mai raggiungere una soluzione stabile.

- **Osservabilità parziale:** In caso di Single Agent RL, si assume che l'agente abbia accesso a tutte le informazioni necessarie per scegliere azioni ottimali; quindi, non esiste un problema di osservabilità parziale. Tuttavia, in Multi Agent RL gli agenti non riescono ad avere un quadro completo dello stato dell'ambiente in quanto ogni agente ha la propria visione dello stato che è sconosciuta agli altri agenti, il che complica l'apprendimento di una policy ottimale.
- **Scalabilità:** Nel contesto del Single Agent RL, il problema della scalabilità è generalmente meno rilevante, poiché l'aumento della complessità del sistema è legato alla dimensione dello spazio di stato e al numero di azioni. Tuttavia, nel Multi-Agent RL, il numero di possibili combinazioni di azioni cresce esponenzialmente con il numero di agenti. In particolare, se si considera un sistema con N agenti, ciascuno con uno spazio di azioni A_i (dove $i = 1, 2, \dots, N$), lo spazio delle azioni congiunte del sistema è dato dal prodotto cartesiano degli spazi di azione individuali:

$$A = \prod_{i=1}^N A_i.$$

Questo porta a una crescita esponenziale del numero di combinazioni di azioni, con conseguenti aumenti nei costi computazionali e nella complessità del sistema.

Gli algoritmi MARL possono essere classificati in base alle informazioni disponibili durante l'addestramento e l'esecuzione delle policy[18]. Durante l'addestramento, esistono due principali modalità :

- **Addestramento decentralizzato:** Ogni agente utilizza solo informazioni locali per addestrare la propria policy.
- **Addestramento centralizzato:** Gli agenti possono accedere alle informazioni di tutti gli altri agenti per addestrare la policy, per esempio tramite l'utilizzo di un controller centrale.

Dopo l'addestramento, la modalità di esecuzione si distingue in:

- **Esecuzione decentralizzata:** Ciascun agente seleziona le sue azioni basandosi solo sulla sua storia locale di osservazioni.
- **Esecuzione centralizzata:** Ciascun agente seleziona le sue azioni basandosi anche sulle informazioni degli altri agenti.

Le modalità appena citate si combinano per definire tre categorie principali di algoritmi MARL, descritti ampliamente in [18, 27, 28]:

- **Fully centralized learning;**
- **Independent learning;**
- **Centralized training with decentralized execution (CTDE);**

2.2.1 Fully Centralized Learning

Negli algoritmi di questa categoria, gli agenti utilizzano un meccanismo di condivisione delle informazioni, spesso orchestrato da un controller centrale. Quest'ultimo permette di condividere dati come le storie delle osservazioni locali degli agenti, modelli dell'ambiente, funzioni di valore o persino le policy degli agenti stessi. Questo approccio offre il vantaggio principale di poter sfruttare lo spazio di osservazione congiunto dell'ambiente. Tale capacità risulta particolarmente utile in ambienti caratterizzati da parziale osservabilità o che richiedono una coordinazione complessa tra gli agenti.

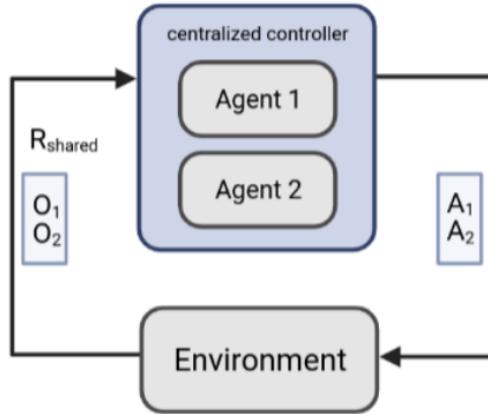


Figura 2.7: Esempio FCL [28].

Tuttavia, l'apprendimento centralizzato spesso non è applicabile per molteplici motivi, come:

- **Reward congiunta:** In alcuni casi, trasformare una ricompensa congiunta in segnali di reward individuali per ogni agente può risultare complicato.
- **Crescita dello spazio d'azione:** La policy centrale deve apprendere sull'insieme combinato delle azioni di tutti gli agenti. Questo spazio d'azione cresce esponenzialmente con l'aumentare del numero di agenti, causando problemi di scalabilità.

- **Distribuzione degli agenti:** In scenari in cui gli agenti sono entità fisiche o virtuali distribuite, la comunicazione con una policy centrale potrebbe non essere fattibile a causa di vincoli di latenza, larghezza di banda o affidabilità della rete.

2.2.2 Indipendent Learning

In questa categoria, ogni agente si occupa autonomamente di sviluppare la propria policy, senza fare affidamento a informazioni o meccanismi condivisi centralmente. Sostanzialmente tutto avviene in modo indipendente tra gli agenti. Questo approccio è particolarmente adatto per scenari in cui non è possibile addestrare o eseguire il controllo centralizzato, come avviene ad esempio nei mercati finanziari. Questo perché, chi opera nel trading spesso non ha informazioni su come si operano gli altri attori o su come le loro azioni influenzano il mercato. Queste influenze sono visibili solo parzialmente e in modo indiretto. In questo contesto, ogni agente addestra la propria policy a livello locale, utilizzando tecniche di apprendimento Single Agent. Un grande vantaggio di questo approccio è la scalabilità: evita infatti la complessità esponenziale che caratterizza l'apprendimento centralizzato e si adatta perfettamente a situazioni in cui gli agenti sono entità fisiche o virtuali distribuite, senza possibilità di comunicare tra loro. Tuttavia, l'apprendimento indipendente non è esente da difficoltà. Ecco alcune delle principali sfide:

- **Mancanza di coordinazione tra gli agenti:** Gli agenti non possono sfruttare alcuna informazione relativa agli altri, né durante l'addestramento delle loro policy né durante la loro esecuzione.
- **Ambiente non stazionario:** Poiché tutti gli agenti si addestrano simultaneamente, l'ambiente percepito da ciascuno di loro cambia costantemente rendendo difficile l'apprendimento.
- **Ambiguità nelle transizioni ambientali:** Gli agenti non riescono a distinguere chiaramente tra cambiamenti stocastici nell'ambiente e cambiamenti causati dalle azioni degli altri agenti, rendendo l'apprendimento meno prevedibile e instabile.

È importante sottolineare che l'apprendimento indipendente non è l'unico approccio per implementare un addestramento decentralizzato. Ci sono altre tecniche che non verranno trattate e che possono essere esplorate per affrontare meglio alcune di queste sfide.

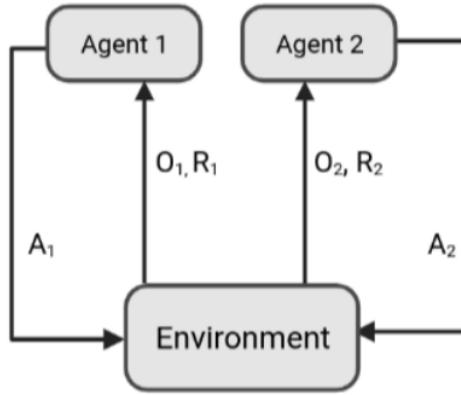


Figura 2.8: Esempio IL [28].

2.2.3 CTDE

Gli algoritmi sopracitati sono caratterizzati da un addestramento centralizzato ed una esecuzione decentralizzata. Durante l'addestramento, l'algoritmo può sfruttare le informazioni condivise da tutti gli agenti per aggiornare le loro policy. Tuttavia, durante l'esecuzione, ogni agente utilizza solo le proprie osservazioni locali per selezionare le azioni, consentendo così un'esecuzione autonoma. Questo approccio combina i vantaggi dell'addestramento centralizzato, che sfrutta informazioni globali, e dell'esecuzione decentralizzata, che rende gli agenti indipendenti. Gli algoritmi CTDE sono spesso basati su una struttura *Actor-Critic*. Una possibile implementazione prevede una rete di critici centralizzata che raccoglie informazioni globali da tutti gli agenti, mentre per ciascun agente viene utilizzata una rete separata di attori. Durante l'addestramento, l'ambiente invia al critico lo stato globale, mentre ciascun agente riceve solo una parte di esso, cioè la sua osservazione parziale. Ad ogni passo temporale t , il critico elabora queste informazioni globali e fornisce un valore, ad esempio la funzione del valore di stato, a tutti gli agenti. In seguito, ogni attore seleziona un'azione in base alla propria policy e la invia all'ambiente. Così facendo, lo stato cambia con la probabilità di transizione $T(s_{t+1}|s_t, a_t)$ e la ricompensa $R^{t+1}(s_t|s_{t+1}, a_t)$ viene calcolata dall'ambiente e restituita sia agli agenti che al critico. Nel processo di aggiornamento, l'attore utilizza la ricompensa ricevuta e le informazioni globali fornite dal critico per migliorare la propria policy, mentre il critico usa esclusivamente la reward per aggiornare il valore dello stato o dell'azione. Durante l'esecuzione, tuttavia, il critico non è più coinvolto; ogni agente utilizza solo le proprie osservazioni locali per prendere decisioni autonome.

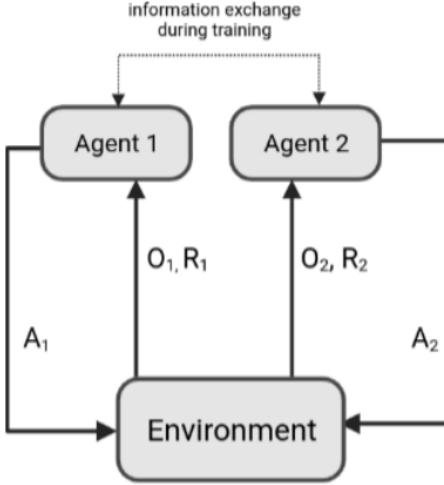


Figura 2.9: Esempio CTDE [28].

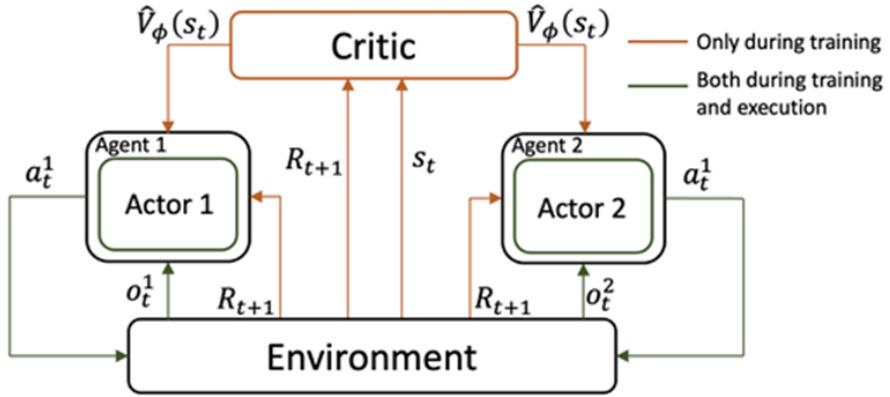


Figura 2.10: CTDE con struttura Actor-Critic [29].

2.3 Algoritmi MADRL

Gli algoritmi MADRL utilizzano delle DNN per rappresentare le policy o le funzioni valore degli agenti invece di utilizzare rappresentazioni tabellari o modelli lineari, tipicamente impiegati negli approcci MARL tradizionali. Questo gli consente di gestire problemi ancora più complessi, e quindi superare il problema della scalabilità discusso nel paragrafo 2.2. Alcuni degli algoritmi principali utilizzati nel MADRL sono:

- Deep Q-Network (DQN);
- Multi-Agent Proximal Policy Optimization (MAPPO);

2.3.1 Deep Q-Network

Il DQN unisce il metodo Q-Learning, precedentemente discusso, alle Deep Neural Network [30, 31]. In questo contesto, l'apprendimento della policy corretta consiste nell'aggiustare i pesi dei neuroni che compongono la rete attraverso la backpropagation. L'aspetto chiave di questa tecnica è l'utilizzo dell' *Experience replay*; questa consiste nel salvare ad ogni time step l'esperienza dell'agente in un dataset D chiamato Replay memory. Così facendo si può effettuare il training attraverso dei mini-batch, che sono dei sottoinsiemi di campioni prelevati randomicamente da D ; L'Experience replay permette:

- Alle esperienze passate di essere usate in più passi di aggiornamento della rete.
- Di interrompere la forte correlazione presente tra esperienze successive riducendo così la varianza tra gli aggiornamenti.

2.3.2 Multi-Agent Proximal Policy Optimization

Analogamente al PPO Single Agent, nel MAPPO vengono apprese: una policy con parametri θ e una value function con parametri ϕ . Queste due funzioni possono essere rappresentate come due reti neurali separate che seguono la struttura CTDE attore-critico precedentemente discussa nel paragrafo 2.2.3 [26, 32]. Infatti, vengono utilizzate una rete attore con parametri θ e una rete critico con parametri ϕ . Ogni agente disporrà della propria rete attore per stimare la funzione di policy utilizzando come input un'osservazione parziale dello stato globale. Ciò produce come output una distribuzione categorica sulle azioni nello spazio discreto delle azioni. Per migliorare ulteriormente la stima del gradiente, possono essere raccolte N traiettorie; Una traiettoria, indicata con τ_j , rappresenta una sequenza di esperienze raccolte da un agente durante un episodio di interazione con l'ambiente. Essa è composta da una serie di tuple (o_t, a_t, r_t, o_{t+1}) , che includono osservazioni (o_t), azioni (a_t), ricompense (r_t) e la successiva osservazione (o_{t+1}) per ciascun istante di tempo t . Così facendo, ciascuna rete attore deve massimizzare la seguente funzione obiettivo:

$$L_{\text{PPO}^i}(\theta^i) = \frac{1}{N_{\text{tr}}} \sum_{j=1}^{N_{\text{tr}}} \mathbb{E}_{\tau_j} \left[L_{t_{\text{PG}}, \text{CLIP}}(\theta^i) + cS[\pi_{\theta^i}(\cdot | o_t^i)] \right], \quad (2.12)$$

dove: \mathbb{E}_{τ_j} rappresenta la media delle esperienze raccolte lungo la traiettoria τ_j ,

$$L_i^{\text{PG}, \text{CLIP}}(\theta^i) = \min \left(r_t^i(\theta^i) \hat{A}_t^i, \text{clip} (r_t^i(\theta^i), 1 - \epsilon, 1 + \epsilon) \hat{A}_t^i \right), \quad (2.13)$$

$$r_t^i(\theta^i) = \frac{\pi_{\theta^i}(a_t^i | o_t^i)}{\pi_{\theta_{\text{old}}^i}(a_t^i | o_t^i)}, \quad (2.14)$$

e \hat{A}_t^i è l'advantage stimato relativo all'agente i-esimo.

Il calcolo dell'advantage richiede l'apprendimento della Value function $\hat{v}_\phi(s_t)$ tramite una rete con critico centralizzato, che prende come input lo stato globale.

$$\hat{A}_t^i = G_t(\tau_j) - \hat{v}_\phi(s_t) \quad (2.15)$$

Tale rete con critico centralizzato ha come *goal* quello di minimizzare la seguente funzione obiettivo:

$$L_{\text{VF}}(\phi) = \frac{1}{N_{\text{tr}}} \sum_{j=1}^{N_{\text{tr}}} \mathbb{E}_{\tau_j} [(\hat{v}_\phi(s_t) - G_t(\tau_j))^2] \quad (2.16)$$

Capitolo 3

Quantizzazione per Deep Neural Network

La *quantizzazione* rappresenta una tecnica fondamentale nel campo dell’ottimizzazione dei modelli, impiegata strategicamente per ridurre la complessità computazionale e migliorare l’efficienza. Questo processo prevede la riduzione della precisione dei valori numerici utilizzati all’interno di un modello, con l’obiettivo finale di ottenere un framework computazionale più efficiente e ottimizzato in termini di risorse. Questo capitolo introdurrà il concetto di quantizzazione, analizzandone i suoi principali metodi.

3.1 Introduzione alla quantizzazione

Nel campo dell’*Internet of things* (IoT) è molto importante controllare l’energia consumata dai dispositivi in quanto questi hanno risorse limitate; Per esempio, i dispositivi IoT, spesso, funzionano a batteria, ciò comporta una limitazione per quel che concerne il loro tempo di operatività; infatti, quando la batteria si scarica il dispositivo non è più in grado di operare e bisogna ricaricarla o sostituirla per permettere al dispositivo di tornare in attività; Tutto ciò influisce sulla manutenzione del dispositivo, comportando implicazioni significative sia dal punto di vista tecnico che economico. Per esempio, un arresto improvviso di un dispositivo può portare alla perdita di dati importanti e all’interruzione delle funzionalità che svolgeva compromettendo l’affidabilità del sistema. Per questi e altri motivi, risulta essenziale in ambito IoT ottimizzare il consumo di energia di tali dispositivi. L’energia consumata da un dispositivo è data dalla somma della *Data energy* e della *Computation energy*; La prima si riferisce all’energia consumata per prelevare i dati e trasportarli nell’unità di processamento, mentre la seconda

è l'energia dovuta alla computazione.

$$Energia totale = Data energy + Computation energy. \quad (3.1)$$

L'energia che ha un impatto più significativo nel calcolo dell'energia consumata è quella relativa ai dati e per ridurla bisogna fare in modo di diminuire il costo dello spostamento dei dati, per esempio con la compressione, oppure riducendo il numero di volte che il dato viene preso da una sorgente ad alto costo (una memoria esterna); Questo si ottiene prelevando il dato dalla memoria esterna ed inserendolo in una SRAM vicina (costo più basso) in modo da non scomodare più la memoria esterna. Mentre per ridurre l'energia dovuta alla computazione si possono utilizzare aritmetiche con bit ridotti.[33] Entrambi i tipi di energia possono essere ridotti tramite la quantizzazione, ovvero una tecnica che permette di passare da una rappresentazione dei dati a precisione maggiore a una rappresentazione dei dati a precisione inferiore [34, 35]. Questa tecnica può essere vista come una funzione che mappa i valori da un dominio dove i dati vengono rappresentati da x_1 bit ad un altro dominio dove i dati vengono rappresentati da x_2 bit. Questa funzione può essere descritta da:

$$F(s, x, z) = Round(s \cdot x) - z = x_{quantized}, \quad (3.2)$$

dove:

- x rappresenta il valore nel dominio originario;
- z rappresenta il *punto zero*, ovvero, l'*offset* che mappa il valore zero del dominio originale con quello del dominio quantizzato;
- *Round* rappresenta il tipo di arrotondamento utilizzato e determina quale livello assegnare al valore originale nel dominio quantizzato. In questo caso l'arrotondamento utilizzato è quello che assegna il valore più vicino. Per esempio, se il valore da arrotondare è 2.51 questo diverrà 3, se invece è 1.12 diverrà 1. Se si esegue la quantizzazione manualmente, senza l'ausilio di librerie che implementano automaticamente il processo, oppure modificando il codice delle librerie stesse, è possibile adottare altre strategie di arrotondamento, tra cui:
 - *Floor* (Arrotondamento per difetto): In cui il valore viene sempre arrotondato verso il basso;
 - *Ceil* (Arrotondamento per eccesso): Il valore viene sempre arrotondato verso l'alto;

- *Troncamento*: I valori decimali vengono semplicemente ignorati.
- s rappresenta lo *scale* (o fattore di scala) ed è calcolato come l'inverso del passo di quantizzazione Δ :

$$\Delta = \frac{1}{s} = \frac{x_{1\max} - x_{1\min}}{2^{x_2-1} - 1} \quad (\text{quantizzazione int}) \quad (3.3)$$

$$\Delta = \frac{1}{s} = \frac{x_{1\max} - x_{1\min}}{2^{x_2} - 1} \quad (\text{quantizzazione unsigned int}) \quad (3.4)$$

Dove $x_{1\max}$ e $x_{1\min}$ sono il valore massimo e minimo del dominio originale, e x_2 è il numero di bit utilizzati nella rappresentazione del dominio quantizzato.

Il passo di quantizzazione definisce la distanza tra due valori nel dominio quantizzato e permette di distinguere due tipi di quantizzazione [36]:

- **Quantizzazione uniforme**: I valori appartenenti al dominio quantizzato sono distribuiti in modo equidistante tra di loro. In breve, Δ rimane costante in tutto il dominio.
- **Quantizzazione non uniforme**: I valori appartenenti al dominio quantizzato sono distribuiti in maniera non equidistante. In breve, Δ non è costante. Uno degli approcci più utilizzati in questa casistica è il quantizzatore logaritmico, in cui lo scale aumenta esponenzialmente.

Il modo in cui i valori del dominio originale vengono mappati in quello quantizzato, permette di distinguere altri due tipi di quantizzazione [34]: *simmetrica* e *asimmetrica*.

Quantizzazione simmetrica Nella quantizzazione simmetrica, il valore quantizzato equivalente a zero nel dominio originale è esattamente zero, in modo che gli intervalli di entrambi i domini risultino simmetrici intorno lo zero. Essendo $z = 0$, questo tipo di quantizzazione riduce la complessità dei calcoli per trovare il valore quantizzato:

$$F(s, x) = \text{Round}(s \cdot x) = x_{\text{quantized}}. \quad (3.5)$$

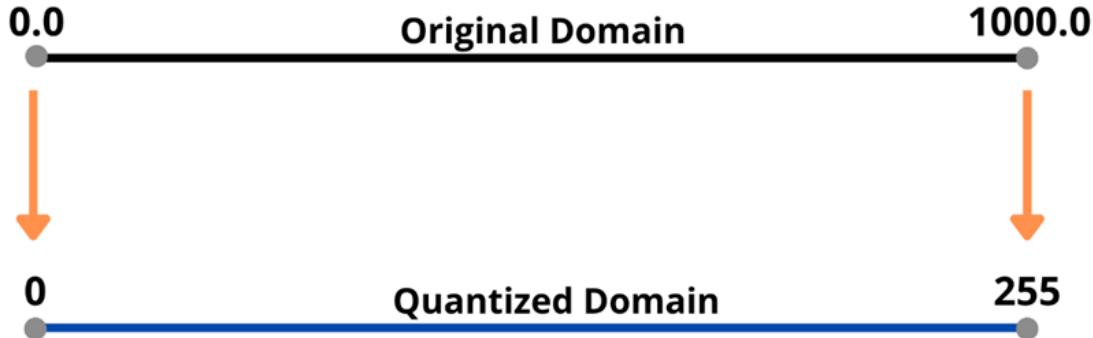


Figura 3.1: Esempio di quantizzazione simmetrica con intervallo quantizzato uint8.

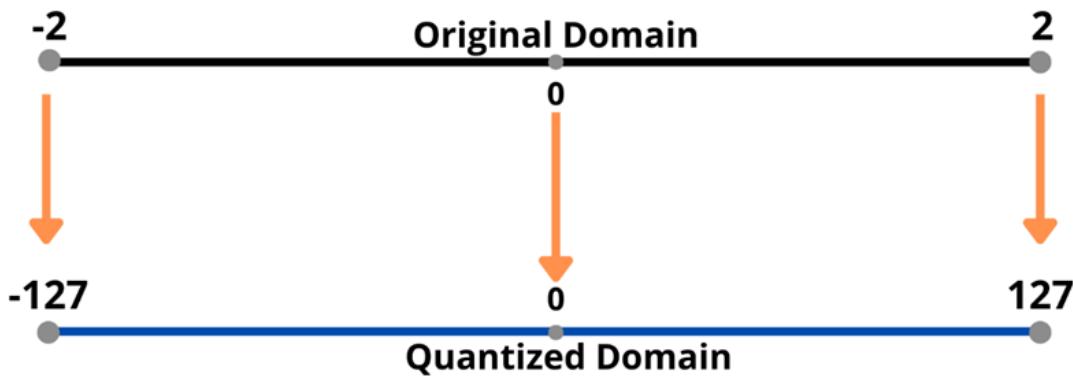


Figura 3.2: Esempio di quantizzazione simmetrica con intervallo quantizzato int8.

Questo tipo di quantizzazione può comportare dei problemi se l'intervallo originale è distribuito in modo non uniforme attorno al punto zero e se si sceglie una rappresentazione non adeguata. Per esempio, considerato un range dei valori in virgola mobile: [0.0, 4000.0] e un dominio quantizzato con 8 bit in formato signed $[-127, -27]$; La parte negativa del dominio quantizzato $[-127, -1]$ rimane inutilizzata, portando così alla perdita di quasi metà dei valori utili all'interno dello spazio quantizzato. In questo caso sarebbe stato più appropriato utilizzare nel dominio quantizzato una rappresentazione in formato unsigned int.

Oppure, considerato un range dei valori in virgola mobile: $[-2, 4]$ e un dominio quantizzato con 8 bit in formato unsigned $[0, 255]$, un altro problema che potrebbe capitare è quello di non poter associare ai valori originali alcun valore nel dominio quantizzato. Nel caso dell'esempio proposto l'intervallo dei valori $[-2, 0[$ del dominio originale non può essere rappresentato in quello quantizzato. In questo caso sarebbe stato più appropriato utilizzare nel dominio quantizzato una rappresentazione in formato signed int o, meglio ancora, una quantizzazione di tipo asimmetrica.

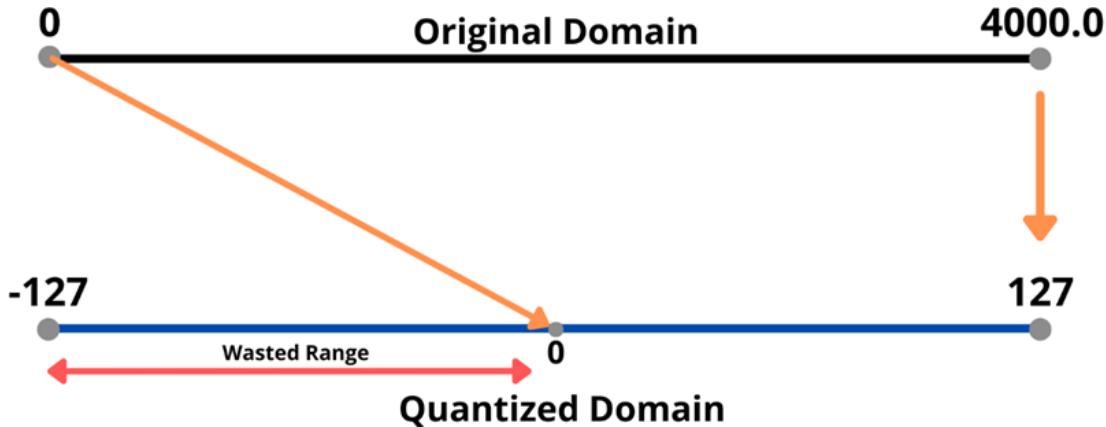


Figura 3.3: Esempio di quantizzazione simmetrica con perdita di intervallo utile $[-127, -1]$.

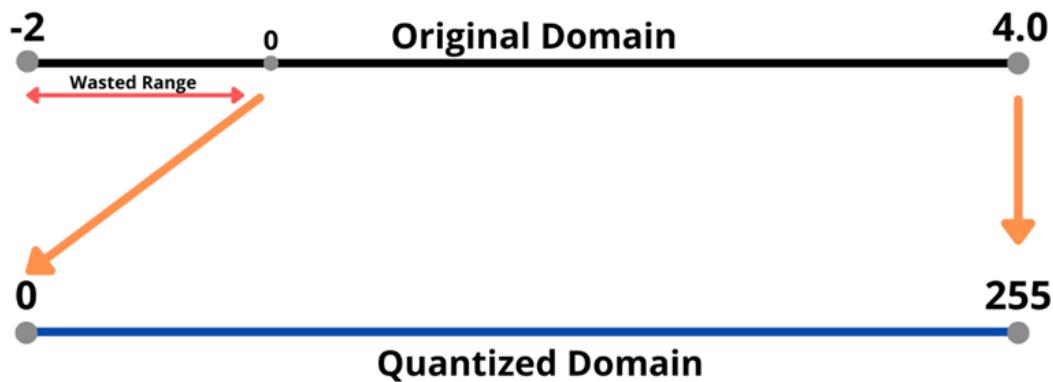


Figura 3.4: Esempio di quantizzazione simmetrica con perdita di valori rappresentabili $[-2, 0]$.

Quantizzazione asimmetrica Nella quantizzazione asimmetrica, al contrario di quella simmetrica, il punto zero ha sempre un valore diverso da zero. Per esempio, se nel dominio originale si ha un intervallo di valori $[-10.0, 1000.0]$ il punto zero in un dominio quantizzato con rappresentazione unsigned int a 8 bit corrisponde a:

$$z = \text{Round}(-s \cdot x_{1_{\min}}) \quad (3.6)$$

In questo caso lo scale vale:

$$\Delta = 1/s = [1000.0 - (-10)]/(2^8 - 1) = 1010/255 = 3.96, \text{ da cui } s = 0.25 \quad (3.7)$$

Sostituendo s e $x_{1_{\min}}$ nella formula di z , si ricava:

$$z = \text{Round}(-0.25 \cdot -10.0) = \text{Round}(2.5) = 3 \quad (3.8)$$

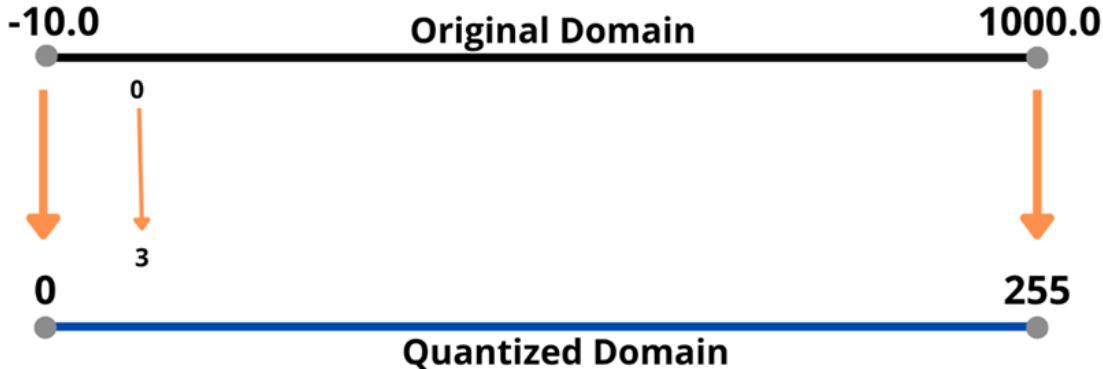


Figura 3.5: Esempio di quantizzazione asimmetrica con intervallo quantizzato in uint8.

Nel caso in cui il dominio quantizzato abbia una rappresentazione signed int a 8 bit, il punto zero corrisponde a:

$$z = \text{Round}(-s \cdot x_{1_{\min}}) - 2^{x-1} \quad (3.9)$$

Anche in questo caso lo scale vale:

$$\Delta = 1/s = [1000.0 - (-10)]/(2^8 - 1) = 1010/255 = 3.96, \text{ da cui } s = 0.25 \quad (3.10)$$

Sostituendo s e $x_{1_{\min}}$ nella formula di z , si ricava:

$$z = \text{Round}(-0.25 \cdot -10.0) - 128 = \text{Round}(2.5) = 3 - 128 = -125 \quad (3.11)$$

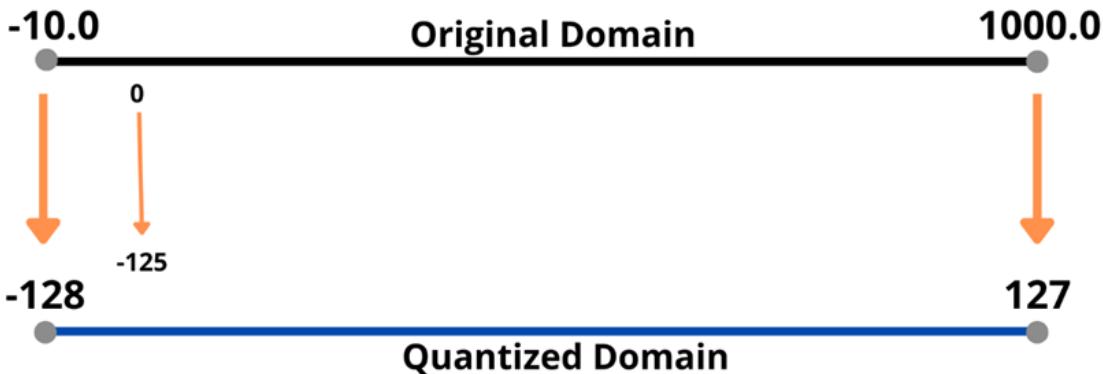


Figura 3.6: Esempio di quantizzazione asimmetrica con intervallo quantizzato in uint8.

In questo tipo di quantizzazione la computazione è più complessa poiché bisogna considerare il punto zero nella formula e memorizzarlo in memoria, però non si hanno problemi di rappresentazione. Ovvero, si può rappresentare qualsiasi intervallo di un

dominio originale, però non è detto che quell'intervallo sia adatto al tipo di quantizzazione adottato, causando un *overhead* computazionale. Per esempio, considerato un range dei valori in virgola mobile: $[-20.0, 20.0]$ è meglio utilizzare una quantizzazione simmetrica invece di quella asimmetrica in quanto più efficiente in termini di utilizzo della memoria e di operazioni computazionali.

3.1.1 Motivazioni e sfide

La quantizzazione permette di ridurre la Computation energy e la Data energy del dispositivo migliorando così l'efficienza computazionale e riducendo il consumo energetico, in quanto:

- Le operazioni aritmetiche, come moltiplicazioni e somme, richiedono meno cicli di clock per essere eseguite dall'unità di elaborazione.
- I dati occupano meno spazio in memoria e richiedono meno energia per essere trasferiti dalla memoria al processore.

Inoltre, la quantizzazione permette di migliorare la scalabilità dell'intero sistema. Infatti, si può sfruttare la migliore efficienza computazionale ottenuta per gestire carichi di lavoro più elevati senza dover investire in hardware aggiuntivo.

I benefici appena discussi comportano un bilanciamento tra accuratezza e precisione, ovvero bisogna trovare per la rappresentazione dei dati, un valore di precisione che mantenga le prestazioni all'interno di un intervallo tollerabile. Un dato in forma quantizzata non può rappresentare le stesse informazioni del dato nella sua forma originale e tale perdita di precisione viene definita come errore di quantizzazione. In generale, questo errore è inversamente proporzionale al numero dei bit utilizzati per rappresentare il segnale; quindi, utilizzare un numero di bit basso per rappresentare i dati introduce un errore più alto rispetto all'utilizzare un numero di bit più elevato. Tuttavia, utilizzare un numero basso di bit migliora l'efficienza computazionale, il consumo energetico e la scalabilità rispetto ad un numero di bit più alto, ma non è detto che l'accuratezza del dispositivo rimanga la stessa in quanto aumenta l'errore. L'*errore di quantizzazione* per un singolo valore x può essere calcolato come:

$$e = x - \hat{x}_i, \quad (3.12)$$

dove x_i corrisponde al valore x dequantizzato, ovvero il valore nel dominio originale ricostruito a partire dal valore quantizzato.



Figura 3.7: Un’immagine quantizzata confrontata con l’originale [34].

In base al segno dell’errore si può capire come si comporta il processo di quantizzazione: Se $e < 0$, significa che il processo sta sovrastimando il valore originale (cioè, il valore dequantizzato è maggiore del valore originale). Se $e > 0$, significa che il processo sta sottostimando il valore originale (cioè, il valore dequantizzato è minore del valore originale) [34]. Per esempio, si consideri il caso con $\Delta_1 = 0.023$ e $x = 0.5$. L’errore di quantizzazione sarà:

$$e_1 = x - \hat{x}_i = \text{Round} \left(\frac{0.5}{\Delta_1} \right) \Delta_1 = 0.5 - 22 \cdot \Delta_1 = 0.5 - 0.506 = -0.006$$

Mentre in un secondo caso con $\Delta_2 = 0.007$ e $x = 0.5$, l’errore di quantizzazione sarà:

$$e_2 = x - \hat{x}_i = \text{Round} \left(\frac{0.5}{\Delta_2} \right) \Delta_2 = 0.5 - 7 \cdot \Delta_2 = 0.5 - 0.049 = 0.001$$

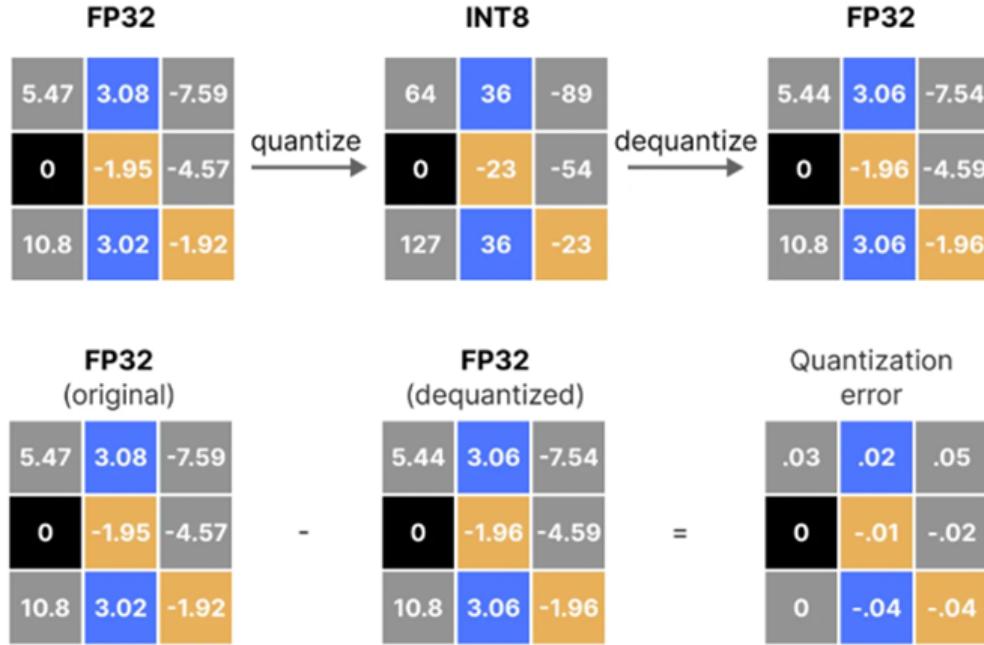


Figura 3.8: Esempio sul calcolo dell'errore di quantizzazione [34].

La sfida relativa al *trade-off* tra precisione ed accuratezza consiste nel capire quanto errore può essere tollerato senza avere una perdita significativa in termini di accuratezza e sfruttare i vantaggi energetici e computazionali introdotti dalla quantizzazione. Ovviamente, la scelta del numero di bit da utilizzare varia da applicazione ad applicazione, in quanto si possono avere esigenze diverse. In applicazioni, come per esempio il video streaming, un leggero degrado della qualità può essere accettabile, mentre in altre applicazioni, come la simulazione scientifica, anche una piccola perdita di precisione potrebbe avere conseguenze significative sui risultati.

Invece, per calcolare l'errore medio di quantizzazione su un set N di valori, si può utilizzare il *Mean Square Error* (MSE) [37]. L'MSE misura la media delle differenze al quadrato tra i valori previsti e i valori target effettivi all'interno del set ed è calcolato come:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (x_i - \hat{x}_i)^2, \quad (3.13)$$

dove:

- N rappresenta il numero totale di valori nel set;
- x_i rappresenta il valore target, nel nostro caso il valore originale;
- \hat{x}_i rappresenta il valore previsto, nel nostro caso il valore dequantizzato,

La presenza del quadrato nella formula (3.13) fa in modo che l'MSE attribuisce un peso maggiore agli errori più grandi, rendendolo sensibile ai valori anomali. In generale, un MSE più basso indica che i valori previsti sono più vicini ai valori target; Viceversa, un MSE più alto indica che i valori previsti sono più lontani dai valori target.

Un altro problema comune durante la quantizzazione è rappresentato dagli *outlier*, ovvero dei valori anomali che si discostano in maniera significativa dagli altri valori [34]. Gli outlier aumentano considerevolmente il passo di quantizzazione Δ , incrementando l'errore di quantizzazione. Per esempio, sia considerato un segnale con valori prevalentemente compresi tra $[-1, 1]$, ma con un outlier di valore 5. Se si include l'outlier nel range della quantizzazione, Δ diventa:

$$\Delta_1 = \frac{5 - (-1)}{2^8 - 1} = \frac{6}{255} = 0.023 \quad (3.14)$$

Mentre se non lo si include, limitando il range a $[-1, 1]$, Δ vale:

$$\Delta_2 = \frac{1 - (-1)}{2^8 - 1} = \frac{2}{255} = 0.007 \quad (3.15)$$

Come già discussso nel paragrafo 3.1, più grande è il Δ , più distanti sono i livelli di quantizzazione, quindi maggiore sarà l'errore di quantizzazione, perché ogni valore x sarà approssimato a un livello discreto più distante. In questo caso, all'interno dell'intervallo $[-1, 1]$, l'errore di quantizzazione e_1 sarà quasi sempre maggiore di e_2 . Nei grafici a seguire, è possibile osservare come nell'esempio appena citato con Δ_1 si raggiungono spesso errori di quantizzazione più elevati rispetto a Δ_2 .

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def quantize_error(data, quantization_step):
5     quantized_data = np.round(data/quantization_step)
6     dequantized_data = quantized_data * quantization_step
7     e = data - dequantized_data
8     return e
9
10 min = -1.0
11 max = 1
12
13 quantization_step_1 = 0.023 # delta 1
14 quantization_step_2 = 0.007 # delta 2
15
16 data = np.linspace(min, max, 1000)
17
18 e1 = quantize_error(data, quantization_step_1)
19 e2 = quantize_error(data, quantization_step_2)
20
21 # Confronto dei valori assoluti degli errori
22 comparison = []
23 for z, x, y in zip(data, e1, e2):
24     x = np.abs(x)
25     y = np.abs(y)
26
27     if x > y:
28         comparison.append((z, 1)) # 1 se e1 maggiore
29     elif x < y:
30         comparison.append((z, -1)) # -1 se e2 maggiore
31     else:
32         comparison.append((z, 0)) # 0 se sono uguali
33
34 positive_points = [z for z, c in comparison if c == 1]
35 negative_points = [z for z, c in comparison if c == -1]
36 zero_points = [z for z, c in comparison if c == 0]
```

Codice 3.1: Codice python esempio quantizzazione - parte 1

```

37 plt.plot(data, e1, label='Error_1_(delta=0.023)')
38 plt.plot(data, e2, label='Error_2_(delta=0.007)')
39 plt.legend()
40 plt.xlabel('Valori_Originali')
41 plt.ylabel('Errore_di_Quantizzazione')
42 plt.title('Confronto_degli_Errori_di_Quantizzazione')
43 plt.grid(True)
44 plt.show()
45
46 plt.scatter(positive_points, np.ones(len(positive_points)), color='b',
             label='error_1>error_2', s=1)
47 plt.scatter(negative_points, -1 * np.ones(len(negative_points)), color='r',
             label='error_2>error_1', s=1)
48 plt.scatter(zero_points, np.zeros(len(zero_points)), color='g', label='error_1==error_2', s=1)
49 plt.axhline(0, color='black', linewidth=1)
50 plt.title('Confronto_tra_gli_Errori_di_Quantizzazione_(Assoluti)')
51 plt.xlabel('Valori_Originali')
52 plt.ylabel('')
53 plt.grid(True)
54 plt.legend()
55 plt.show()

```

Codice 3.2: Codice python esempio quantizzazione - parte 2

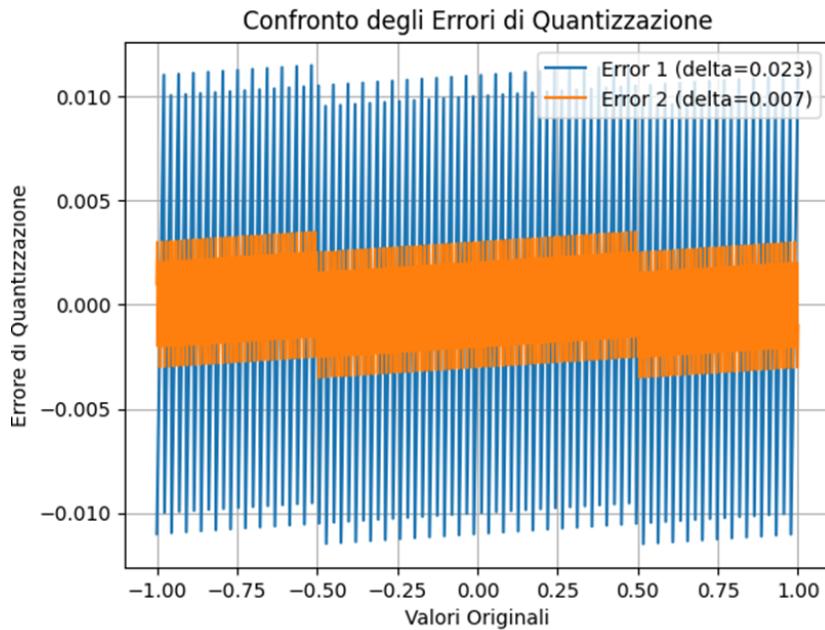
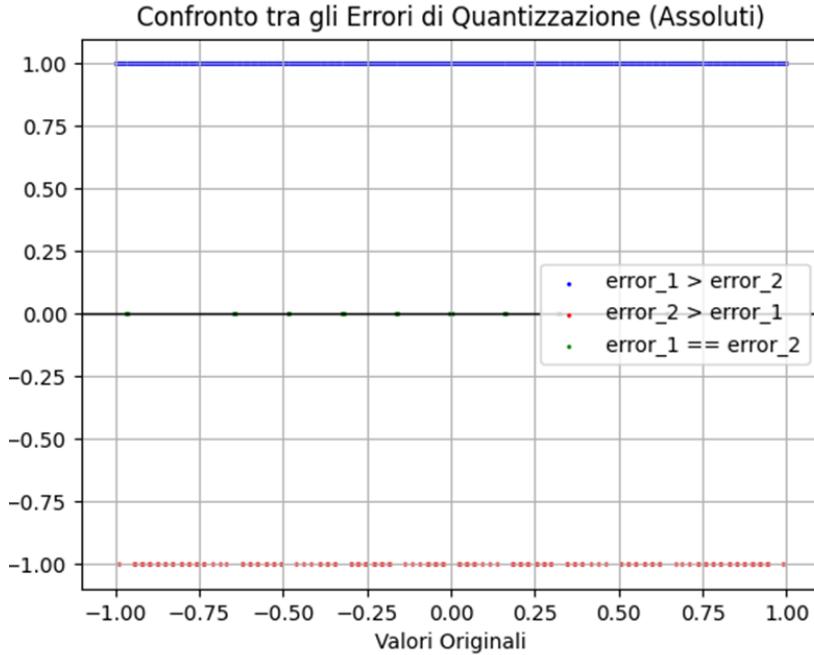


Figura 3.9: Errori di quantizzazione ottenuti con Δ_1 e Δ_2 .

Figura 3.10: Frequenza errori di quantizzazione con Δ_1 e Δ_2 .

Per affrontare il problema degli outlier, si può limitare il range di valori quantizzabili utilizzando una tecnica chiamata *clipping* [34]. Con questa tecnica si stabiliscono un limite massimo e un limite minimo accettabile, ad esempio $\min = -1$ e $\max = 1$. Ogni valore che supera questi limiti viene *troncato* secondo le seguenti regole:

$$x = \begin{cases} \min & \text{se } x < \min \\ \max & \text{se } x > \max \\ x & \text{altrimenti} \end{cases}$$

3.2 Quantizzazione nelle Deep Neural Network

La Quantizzazione può essere applicata anche alle DNN e godere dei vantaggi descritti nel paragrafo 3.1.1; Si dà il caso che le DNN, includono architetture complesse come le Convolutional Neural Networks (CNN), le Recurrent Neural Networks (RNN) e le Fully Connected Networks (FCL); Queste presentano un numero estremamente elevato di parametri (pesi e bias) che devono essere memorizzati e manipolati durante il processo di addestramento e inferenza. Questi parametri sono solitamente rappresentati con una rappresentazione a virgola mobile a 32 bit. Più è alto il numero dei parametri più alto è il numero di risorse computazionali che servono per gestirli, sia in termini di memoria che in termini di potenza di calcolo. Questo diventa un problema, poiché gli hardware

utilizzati per addestrare ed eseguire le DNN hanno risorse limitate come la memoria VRAM delle GPU. In tali contesti, la quantizzazione viene impiegata per evitare di saturare la memoria dell'hardware, ottimizzando l'uso delle risorse disponibili senza compromettere significativamente le prestazioni del modello. Inoltre, la quantizzazione permette di accelerare anche i tempi di esecuzione, però se il modello è troppo semplice le operazioni di quantizzazione/dequantizzazione eseguite potrebbero introdurre un overhead significativo annullando tale beneficio [38, 39].

In generale i passaggi da eseguire sono i seguenti:

1. I parametri e le attivazioni del modello vengono convertiti da una rappresentazione a virgola mobile a 32 bit a una rappresentazione a precisione inferiore, utilizzando numeri interi a 8 bit. Per capire la differenza si pensi che utilizzando una rappresentazione virgola mobile a 32 bit si possono rappresentare circa 4 miliardi di numeri mentre utilizzando una rappresentazione intera a 8 bit solo 256. Questa conversione può essere realizzata tramite due diverse tecniche possono essere implementate facilmente attraverso la libreria PyTorch: la *Quantization Aware Training* (QAT) e la *Post-Training Quantization* (PTQ);
2. Una volta ottenuto il modello quantizzato, si devono quantizzare i dati in input prima che questo esegui un inferenza;
3. Prima di ottenere l'output, durante la fase di inferenza, l'uscita dell'ultimo layer spesso deve essere dequantizzata, in quanto non tutte le operazioni sono permesse nel dominio quantizzato, come per esempio la *Softmax*.

3.2.1 Post-Training Quantization

La PTQ applica la riduzione della precisione dopo l'addestramento del modello e può avvenire in maniera statica o dinamica [40, 41].

Quantizzazione dinamica

Nella quantizzazione dinamica si prende il modello da quantizzare già addestrato (in forma pretrained) e si quantizzano solo i pesi lasciando invece le attivazioni in virgola mobile. Ovvero, i pesi del modello sono quantizzati prima dell'inferenza, mentre le attivazioni sono quantizzate dinamicamente durante l'inferenza, il che significa che la loro scala e il loro punto zero sono calcolati durante l'esecuzione.

Quantizzazione statica

Nella quantizzazione statica si prende il modello da quantizzare già addestrato (in forma pretrained) e si quantizzano sia i pesi che le attivazioni prima dell'inferenza. Questo tipo di quantizzazione è caratterizzata da quattro procedure particolari:

1. **Fused Step** : È una procedura non obbligatoria poiché non sempre possibile, in cui le attivazioni vengono fuse con i *layer* precedenti. Questa fusione implica la combinazione di operazioni come convoluzioni e attivazioni (ad esempio, ReLU) in un singolo passaggio di calcolo, e consente di:

- **Ridurre la latenza**: Evitando operazioni separate, il modello può eseguire più velocemente, riducendo i tempi di inferenza;
- **Minimizzare l'overhead di memoria**: La fusione implica che non è necessario salvare le attivazioni intermedie nella memoria, risparmiando spazio e riducendo il carico della larghezza di banda di memoria;
- **Ottimizzare l'uso dell'hardware**: Riduce il numero di letture e scritture di dati, sfruttando meglio le risorse hardware (ad es., CPU o acceleratori come GPU).

2. **Configuration Step** : Questa è una procedura fondamentale che prepara il modello alla quantizzazione. In questa fase, si effettua:

- **Inserimento degli stub**: Dei layer particolari che una volta inseriti nel modello si occupano di quantizzare (QuantStub) e dequantizzare (DequantStub) i suoi input e i suoi output.
 - *QuantStub*: Durante la fase di calibrazione si comporta come un osservatore che raccoglie informazioni sui dati di input, in modo da determinare il modo in cui mappare i valori. Durante la fase di conversione del modello, viene poi sostituito con un'operazione di quantizzazione;
 - *DequantStub*: Durante la fase di calibrazione si comporta come una matrice identità, ovvero non dequantizza i dati ma li lascia invariati. Durante la fase di conversione del modello viene poi sostituito con un'operazione di dequantizzazione.
- **Scelta dei parametri di configurazione**:
 - *Observer*: Gli osservatori, o observer, hanno il compito di registrare i valori minimi e massimi dei tensori in arrivo. Questi dati vengono poi

utilizzati per calcolare i parametri necessari alla quantizzazione, come scale e offset;

- *Dtype*: Specifica il tipo di formato da utilizzare per la quantizzazione. È possibile scegliere tra `quint8` e `qint8`;
- *Qscheme*: Determina lo schema di quantizzazione da utilizzare. Come, per esempio, quello per tensore o quello per canale;
- *Reduce_range*: Se impostata su `TRUE`, riduce l'intervallo del tipo di dati quantizzato di 1 bit. Questo migliora l'efficienza ma potrebbe far perdere precisione;
- *Quant_min* e *Quant_max*: Definiscono i valori minimo e massimo di quantizzazione.

Per la configurazione di questi parametri, la libreria PyTorch mette a disposizione una serie di configurazioni predefinite in base al tipo di backend utilizzato, tuttavia, gli sviluppatori hanno anche la possibilità di creare qualcosa di personalizzato, permettendogli di adattare la quantizzazione alle proprie esigenze. In totale ci sono quattro tipi di backend:

- **Fbgemm/x86**: Configurazioni pensate per l'architettura x86. Eseguono una quantizzazione per canale (del layer). Questo significa che ogni canale del tensore ha una propria scala e uno zero-point, il che porta a migliorare la precisione ma aumenta la complessità di calcolo.
- **Qnnpack**: Configurazione pensata per le architetture ARM (tipica nei dispositivi mobili). Esegue la quantizzazione per tensore (uniforme per tutto il tensore), che rende i calcoli più semplici ma meno precisi.
- **Onednn**: Configurazione pensata per CPU, con ottimizzazioni specifiche per hardware Intel. Supporta sia la quantizzazione per tensore che per canale, offrendo un buon equilibrio tra precisione e prestazioni.

3. **Calibration Step** : Procedura obbligatoria in cui il modello viene calibrato su un set di dati, in modo tale da permettere agli observer di calcolare parametri fondamentali come il punto zero e la scala.
4. **Convert Step** : In questa fase il modello viene effettivamente quantizzato. Durante questo passaggio, i dati raccolti nella fase di calibrazione vengono utilizzati per applicare la quantizzazione ai pesi e alle attivazioni del modello. Inoltre, come già accennato nel punto 2, viene effettuata la sostituzione degli stub layer.

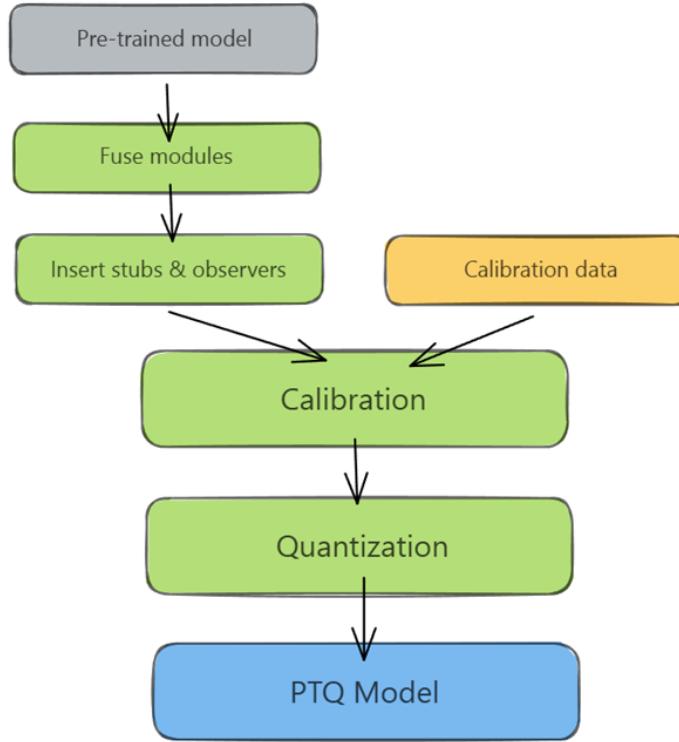


Figura 3.11: Step principali della quantizzazione post training statica [40].

DIFFERENZE TRA QUANTIZZAZIONE POST TRAINING STATICÀ E DINAMICA

Complessità di implementazione A livello di complessità di implementazione, la quantizzazione dinamica è più semplice da implementare rispetto quella statica in quanto prevede meno procedure per essere applicata.

Utilità La quantizzazione dinamica è particolarmente utile in scenari in cui la distribuzione dei dati di input può variare significativamente e non può essere facilmente catturata da un singolo set di dati rappresentativo, come invece avviene nella quantizzazione statica. Mentre la quantizzazione statica è utile in scenari in cui i dati sono stabili e cui distribuzione può essere facilmente catturata nella fase di calibrazione. Sotto questo punto di vista la quantizzazione dinamica risulta più flessibile.

Prestazioni La quantizzazione statica presenta un overhead computazionale ridotto in quanto le scale e gli zero point sono calcolati e fissati, risultando più veloce durante l’inferenza rispetto quella dinamica in cui gli scale e gli zero point sono calcolati durante l’esecuzione.

In conclusione, la quantizzazione statica è preferibile quando si ha bisogno di prestazioni più veloci e stabili e quando i dati non cambiano molto nel tempo. Mentre la quantizzazione dinamica è utile quando si ha variabilità nei dati e il modello deve adattarsi in tempo reale, in cambio di una riduzione delle prestazioni a causa di un overhead maggiore durante l'inferenza.

3.2.2 Quantization Aware Training

La QAT è una tecnica che simula gli effetti della quantizzazione durante la fase di addestramento e fa in modo che il modello si adatti agli effetti della quantizzazione in modo da mantenere alte le prestazioni anche dopo la quantizzazione effettiva. La Quantization aware training è caratterizzata dalle stesse procedure introdotte dalla Quantization post-training statica; tuttavia, la fase di calibrazione è sostituita da una fase di *finetuning*; Durante questa fase, quando il modello esegue una *forward* si introducono delle operazioni chiamate *FakeQuant*, che permettono di emulare la quantizzazione dei pesi e delle attivazioni; questi, infatti, non vengono trasformati veramente ma solo in maniera fittizia, in quanto il loro scopo è quello di abituare il modello senza influire sul calcolo dei gradienti [40, 42].

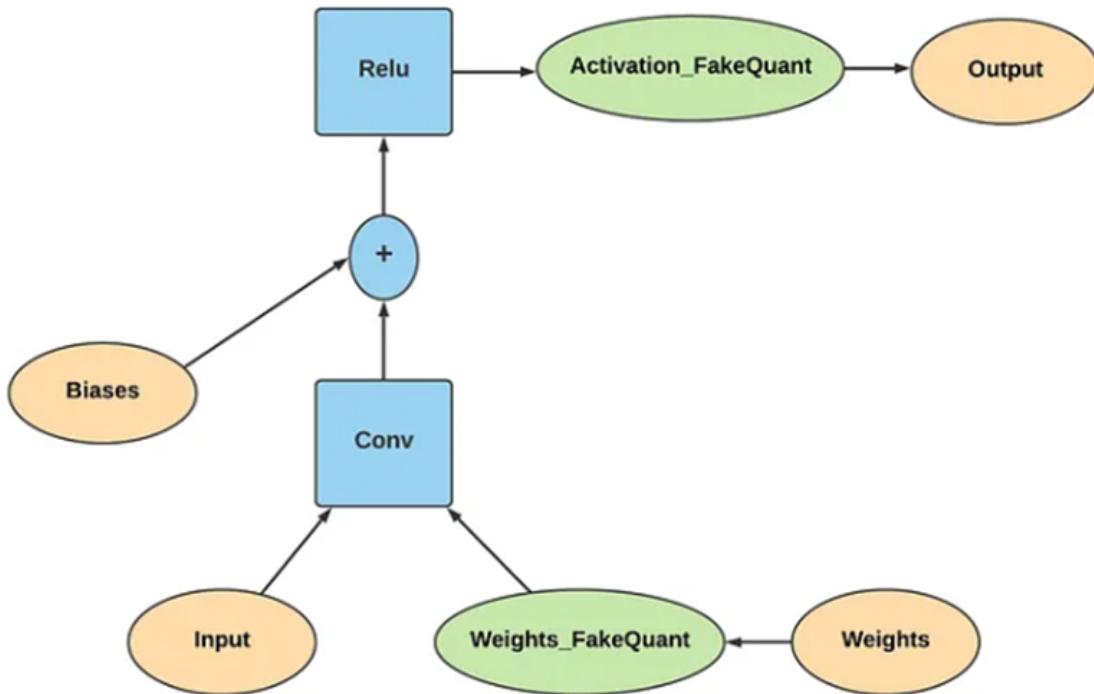


Figura 3.12: Esempio di applicazione dei FakeQuant [43].

Infatti, durante il *backward pass* (backpropagation), i gradienti vengono calcolati normalmente come se i pesi e le attivazioni non fossero mai stati quantizzati. Una volta che i gradienti sono stati calcolati, si procede con l'aggiornamento dei pesi originali. Questo è importante perché la quantizzazione introduce degli errori nei calcoli che possono influenzare negativamente il calcolo dei gradienti e di conseguenza l'aggiornamento dei pesi.

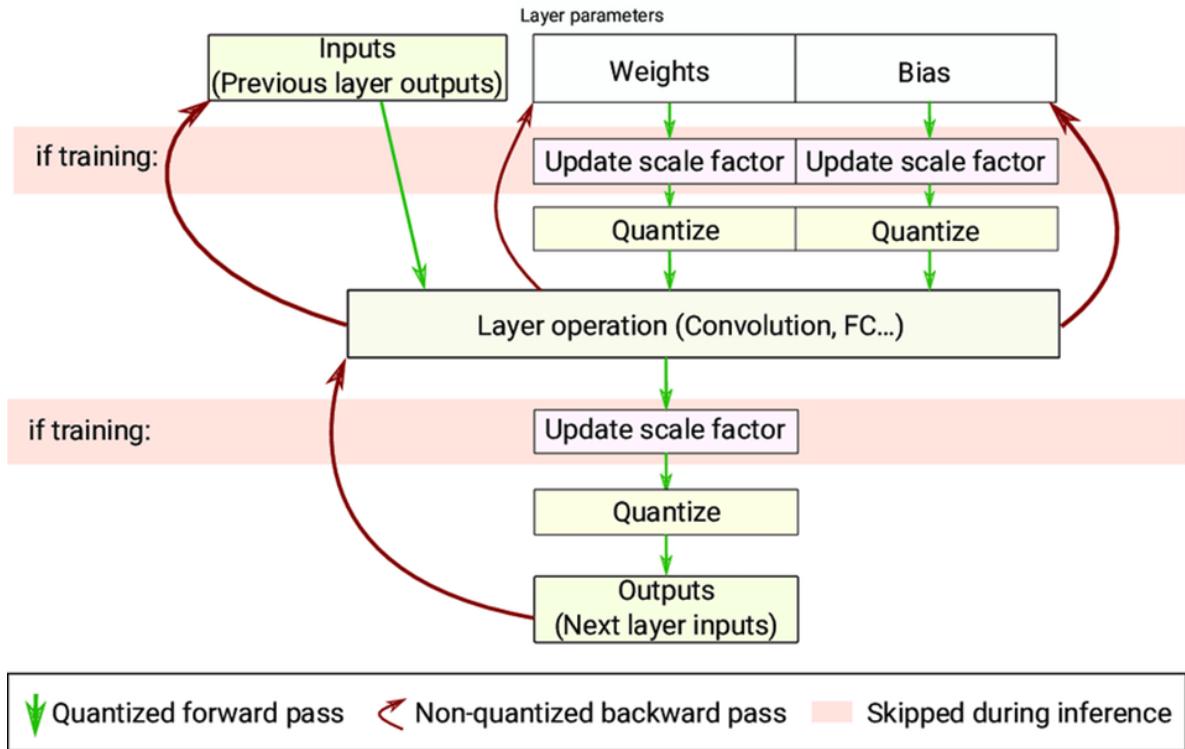


Figura 3.13: Illustrazione della QAT durante la forward e la backpropagation [42].

La quantizzazione introduce una discrepanza tra i valori continui e quelli quantizzati, che può essere non lineare e difficile da derivare. Inoltre, il gradiente per definizione rappresenta la sensibilità di una funzione rispetto a una piccola variazione nei suoi parametri, e la quantizzazione crea un *salto* anziché una variazione continua, rendendo il calcolo del gradiente poco affidabile [44].

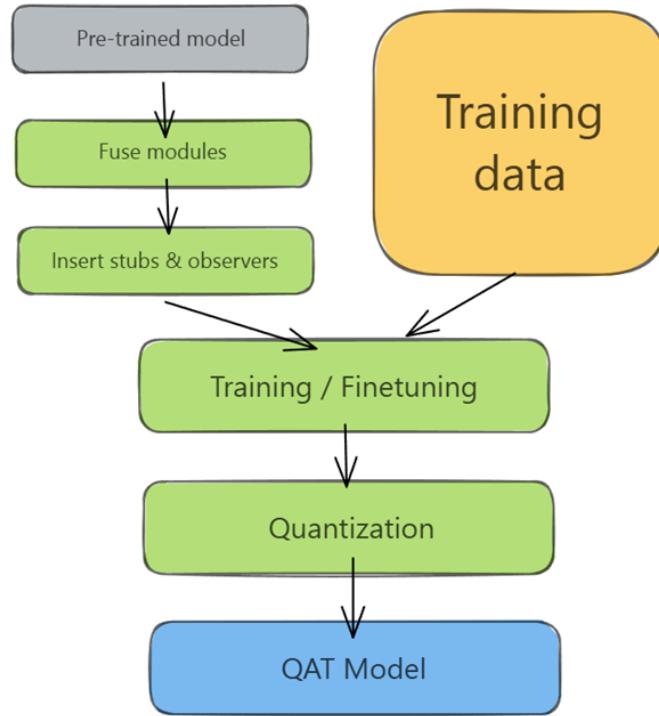


Figura 3.14: Step principali della quantizzazione aware training [40].

3.2.3 Confronto tra PTQ e QAT

Il confronto tra PTQ e QAT è essenziale per comprendere le differenze tra i due approcci e capire quale sia il più adatto in base ai requisiti specifici del modello o dell'applicazione.

Complessità di implementazione A livello di complessità di implementazione la PTQ è più facile da implementare in quanto non richiede una fase *onerosa* di finetuning come nella QAT.

Utilità e prestazioni La PTQ è utile quando l'obiettivo è ottimizzare la memoria e l'efficienza senza necessità di mantenere una precisione ottimale. D'altro canto, se l'obiettivo è mantenere un'alta precisione si utilizza la QAT a scapito di un implementazione più complessa e onerosa.

Limiti Uno dei principali limiti della Post-Training Quantization è rappresentato dal processo di calibrazione dei dati. Una calibrazione non ottimale, ad esempio eseguita con un set di dati poco rappresentativo o di dimensioni insufficienti, può compromettere significativamente le prestazioni, causando una perdita di precisione. Per quel che

concerne la Quantization Aware Training il limite principale è legato alla fase di fine-tuning. Questo processo richiede un riaddestramento del modello, e quindi risulta necessario individuare i giusti iperparametri per stabilizzare il training. Questo rende la QAT meno accessibile rispetto alla PTQ, soprattutto in ambienti con risorse e tempo limitato.

In conclusione, quando la velocità e la facilità di implementazione sono prioritari e non sono necessarie performance perfette si utilizza la PTQ; Mentre in scenari in cui la precisione è essenziale e non ci sono limiti sulla velocità di implementazioni e sulle risorse utilizzate si utilizza la QAT.

3.3 Quantizzazione in MADRL

In ambiente MADRL, ci troviamo spesso a gestire situazioni in cui molti agenti devono interagire con l'ambiente, prendere decisioni rapide e, allo stesso tempo, rispettare vincoli di risorse hardware, come memoria e potenza computazionale. Come già introdotto nel paragrafo 3.1, la quantizzazione del modello permette di ridurre i tempi di esecuzione, la memoria occupata e l'energia consumata. In un contesto MADRL garantire una collaborazione energeticamente vantaggiosa, veloce e performante tra gli agenti può fare la differenza, in quanto permette agli sviluppatori di testare molto più velocemente nuove idee o configurazioni.

Capitolo 4

Metodologia

Alla base del lavoro svolto in questa tesi c'è l'implementazione di una soluzione basata sul paradigma MADRL per l'apprendimento di protocolli *Medium Access Control* (MAC), specificamente progettata per gli ambienti industriali di prossima generazione. Tuttavia, sebbene in quest'ambito le soluzioni basate su MARL dimostrino prestazioni promettenti nelle simulazioni, la loro implementazione pratica presenta sfide significative, in quanto l'esecuzione delle DNN è limitata dalle scarse risorse dei dispositivi che operano in ambienti industriali (IoT device). Questa tesi si concentra sull'utilizzo delle principali tecniche di quantizzazione per alleggerire le Deep Neural Networks e adattarle all'esecuzione su dispositivi IoT, senza compromettere significativamente le prestazioni del modello. In particolare, in questo capitolo, si esploreranno i seguenti aspetti fondamentali:

- **Communication System Model** scelto per la soluzione proposta.
- **Framework MADRL adottato:** Verrà descritta la struttura del framework adottato e, verranno descritte le procedure di training e di testing, e policy DNN utilizzate.
- **KPI di comunicazione:** Verranno discussi i *Key Performance Indicator* (KPI) utilizzati per monitorare l'efficacia della soluzione proposta.
- **Applicazione della Quantizzazione:** Verrà discussa l'applicazione di diverse tecniche di quantizzazione per ottimizzare la DNN proposta, come Post-Training Static Quantization, Post-Training Dynamic Quantization e Quantization-Aware Training.

4.1 Communication System Model

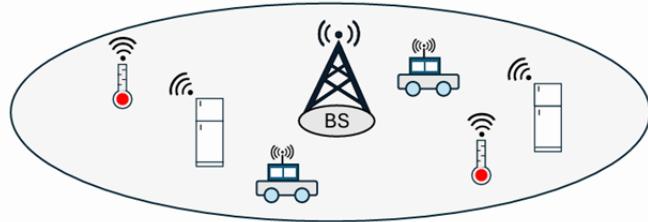


Figura 4.1: Rappresentazione ad alto livello del modello. [45]

Il *Communication System Model* adottato è composto da una Base station (BS) e da device IoT (IoTD) [45].

- **Base Station:** La BS si occupa di gestire l'user plane e il control plane del sistema, garantendo l'invio e la ricezione di messaggi di segnalazione e dati.
- **IoTD:** Gli N IoTD serviti dalla BS sono dotati di un *buffer* FIFO (first-in-first-out) per memorizzare i *Packet Data Unit* (PDU) da trasmettere. Lo stato del buffer al tempo t di un dispositivo $i \in N$ è rappresentato come $b_i^t = \{0, 1, 2, \dots, P\}$, dove P indica la capacità massima del buffer. Inizialmente ($t = 0$) il buffer può essere pieno, contenendo P PDU, o vuoto. In questo scenario consideriamo il buffer pieno;

Gli IoTD operano in modalità *Time Division Multiple Access* (TDMA), con t che indica l'indice dello slot temporale. Ad ogni slot temporale t , gli IoTD eseguono le seguenti operazioni:

- **Piano di controllo:** Ogni IoTD può inviare un messaggio di controllo alla BS e riceverne uno in risposta.
- **Piano dati:** Ogni IoTD può trasmettere la prima PDU nel buffer o eliminarla. La trasmissione di una nuova PDU è subordinata all'eliminazione della precedente.

Il TDMA può generare collisioni quando più dispositivi tentano di trasmettere contemporaneamente nello stesso slot temporale t . Questo problema è molto diffuso nei sistemi con più dispositivi, poiché è altamente probabile che nello stesso momento il buffer di più dispositivi non sia vuoto.

4.1.1 Traffic model

Le PDU negli IoTD vengono generate in base a un modello di traffico che può essere:

- *Periodico*, in cui le PDU vengono prodotte a intervalli regolari.
- *Aperiodico*, in cui le PDU vengono generate in modo irregolare.

4.2 Framework MARL adottato

Il protocollo MAC è fondamentale per coordinare e controllare il modo in cui i dispositivi accedono al canale di comunicazione, riducendo al minimo le collisioni e ottimizzando l'uso del mezzo [45]. In particolare, il protocollo deve:

- Garantire uno scambio ottimale di messaggi di segnalazione tra IoTD e BS;
- Consentire agli IoTD di accedere al canale dati senza generare collisioni.
- Assicurare che ogni PDU venga eliminata dal buffer una volta che è stata correttamente ricevuta dalla BS, rendendo disponibile il buffer per nuove PDUs.

Per raggiungere tale obiettivo la soluzione proposta sfrutta tecniche di RL per addestrare un protocollo MAC sub-ottimale che massimizza l'efficienza del sistema, cioè, massimizzare il numero di PDU ricevuti correttamente dalla BS, riducendo al minimo il tempo necessario per farlo.

La BS adotta un protocollo non appreso e opera esclusivamente nel piano di controllo. In particolare, per ogni slot temporale t , la BS invia un messaggio di controllo $m_i^t \in \{0, 1, 2\}$ a ciascun IoTD. Nel dettaglio:

- Se una PDU inviata da un IoTD i è stata ricevuta correttamente nello slot temporale precedente $t - 1$, la BS invia $m_i^t = 2$, che rappresenta un messaggio di ACK.
- Inoltre, la BS verifica se durante lo slot $t - 1$ sono state ricevute richieste di accesso da parte degli IoTD. In caso positivo, invia $m_i^t = 1$, ovvero un messaggio di concessione di accesso, a uno degli IoTD che ha inviato una richiesta, scelto casualmente.
- Altrimenti, la BS invia $m_i^t = 0$.

Gli IoTD, invece, vengono modellati come agenti RL e il problema di apprendimento del protocollo viene formulato come un MPOMDP, modellato come segue. Ogni IoTD $i \in \{1, 2, \dots, N\}$ ha la propria policy stocastica $\pi_i : O \rightarrow \Delta_i(A)$, dove O è il suo spazio delle osservazioni, A è il suo spazio delle azioni e Δ_i è il suo spazio delle probabilità. A ogni istante temporale t , ogni IoTD calcola la propria osservazione e utilizza la propria policy stocastica per ottenere la distribuzione di probabilità su tutte le azioni $a_i^t \in A$. Successivamente, esegue un'azione a_i^t , estratta dalla distribuzione di probabilità.

L'azione $a_i^t = (a_{(i,u)}^t, a_{(i,s)}^t) \in A$ coinvolge sia il piano dati che il piano di controllo. Nello specifico:

- $a_{(i,u)}^t = \{0, 1, 2\} \in A$ rappresenta l'azione del piano dati:
 - $a_{(i,u)}^t = 1$: L'IoTD trasmette la prima PDU nel suo buffer (se presente);
 - $a_{(i,u)}^t = 2$: L'IoTD elimina la prima PDU nel buffer;
 - $a_{(i,u)}^t = 0$: L'IoTD non esegue alcuna azione.
- $a_{(i,s)}^t = \{0, 1\} \in A$ rappresenta l'azione del piano di controllo:
 - $a_{(i,s)}^t = 1$: L'IoTD invia una richiesta di accesso;
 - $a_{(i,s)}^t = 0$: L'IoTD non trasmette alcun messaggio di segnalazione.

Per quanto riguarda l'osservazione, ogni IoTD è dotato di una memoria interna che memorizza la cronologia passata per affrontare il problema della parziale osservabilità, definita come:

$$o_i^t = (b_i^t, c_i^{t-1}, c_i^{t-2}, \dots, c_i^{t-M}), \quad (4.1)$$

dove

$$c_i^{t-k} = (b_i^{t-k}, a_{i,u}^{t-k}, a_{i,s}^{t-k}, m_i^{t-k}) \quad (4.2)$$

La parziale osservabilità o_i^t contiene le informazioni disponibili dal dispositivo i relative allo slot temporale $t - k$ con $k \in \mathcal{M} = \{1, \dots, M\}$, dove M rappresenta la lunghezza della memoria. Ad esempio:

- $M = 1$: Gli IoTD prendono decisioni basate solo sulle informazioni relative allo slot precedente $t - 1$;
- $M = 2$: Gli IoTD considerano una prospettiva più a lungo termine, includendo anche lo slot $t - 2$.

Quindi, è importante notare che l'azione effettuata nel piano dati è una funzione della sequenza di operazioni effettuate su k slot temporali. Ad esempio, con $M = 2$, l'azione

effettuata al tempo t può tener conto del fatto che al tempo $t - 2$ l'IoTD ha inviato una richiesta di accesso alla BS e al tempo $t - 1$ ha ricevuto una concessione di accesso. In parole povere, il valore della memoria M influenza le prestazioni del protocollo appreso.

4.2.1 Procedure di training e testing

Per generare i protocolli MAC, gli agenti (gli IoTD) eseguono una fase di addestramento [45]. In questa fase, gli IoTD e la BS interagiscono per diversi episodi, ciascuno della durata massima di t_{max} slot temporali. Ad ogni slot temporale t di ciascun episodio, la stazione base fornisce agli agenti un valore di reward globale R^{t+1} come conseguenza dell'azione congiunta $\mathbf{a}^t = (a_1^t, a_2^t, a_3^t, \dots, a_i^t)$.

L'obiettivo di ciascun agente è apprendere la policy ottimale π_i^* , che rappresenta il protocollo MAC appreso e definisce le migliori azioni da intraprendere per accedere al canale di comunicazione. Ogni funzione di policy π_i è una DNN parametrizzata dai pesi θ_i . Pertanto, l'obiettivo finale è ottenere il set ottimale di pesi $\boldsymbol{\theta}^* = (\theta_1^*, \theta_2^*, \dots, \theta_N^*)$. Il vettore di policy ottimali $\boldsymbol{\pi}^* = (\pi_1^*, \pi_2^*, \dots, \pi_N^*)$, che massimizza la ricompensa cumulativa, viene ottenuto adottando l'algoritmo MAPPO.

Una volta conclusa la fase di addestramento, nella fase operativa, ogni agente sfrutterà la politica ottimale π_i^* per eseguire le operazioni corrette nei piani dati e controllo. Nello specifico, ad ogni istante di tempo t , ciascun agente i calcola la propria osservazione o_i^t . Successivamente, fornisce o_i^t come input alla rete di policy π_i^* per calcolare l'inferenza della rete.

Nella tabella seguente sono riportate le condizioni di training.

Parametro	Simbolo	Valore
Numero di dispositivi IoT	N	2
Numero di PDU da trasmettere	P	2
Durata massima dell'episodio	t_{\max}	32
Lunghezza della memoria	M	{1,2,3,4}
Tasso di errore sul blocco di trasmissione (BLER)	-	0.01
Numero di simulazioni	N_{sim}	1000
Condizione di buffer iniziale	case	Full buffer
Discount factor	γ	0.99
Reward function parameter	ρ	3
Learning rate - actor	L_r	$3 * 10^{-4}$
Learning rate - critic	L_c	10^{-3}
Numero di episodi per update	N_{ep}	20
Totale episodi di training	N_{tr}	10K
Clipping value	ϵ	0.2
Entropy coefficient	c	0.01

Tabella 4.1: Parametri del training

Nella fase di evaluation, quasi tutti i parametri definiti precedentemente rimangono invariati, ad eccezione di t_{\max} che è impostato su 300.

4.2.2 Policy DNN

La struttura del modello DNN che rappresenta la policy implementata in ciascun IoT è composta da un *input layer*, due *hidden layer* e un *output layer*.

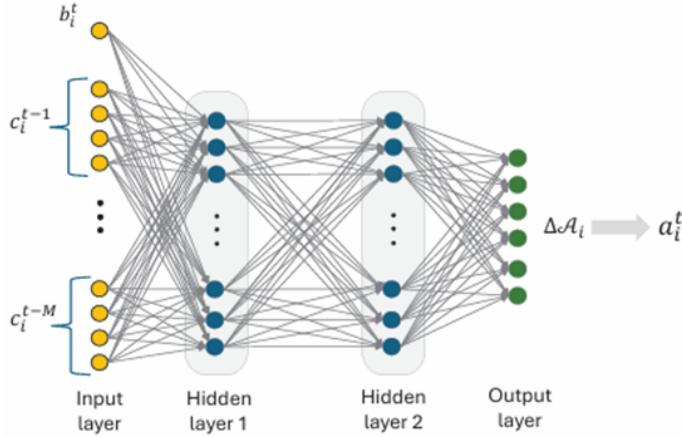


Figura 4.2: Struttura del modello DNN implementato [45].

```

1 import torch.nn as nn
2
3 actor = nn.Sequential(
4     nn.Linear(state_dim, number_of_hidden_nodes),
5     nn.Tanh(),
6     nn.Linear(number_of_hidden_nodes, number_of_hidden_nodes),
7     nn.Tanh(),
8     nn.Linear(number_of_hidden_nodes, action_dim),
9     nn.Softmax(dim=-1)
10 )

```

Codice 4.1: Codice python rappresentante la struttura del modello.

Dalla definizione di o_i^t riportata nel paragrafo 4.2 si deduce facilmente che l'input layer è composto da un numero di neuroni pari a $N_{in} = 4M + 1$. Di conseguenza è facile intuire che all'aumentare della memoria (M) aumenta il numero totale di parametri della rete neurale, portando ad una maggiore complessità computazionale. Invece, ciascun hidden layer è stato progettato per includere 64 neuroni. Infine, per quanto riguarda l'output layer, il numero di neuroni è dato dalla combinazione delle possibili azioni che si possono effettuare nel piano di controllo e nel piano utenti, ovvero:

$$N_{\text{out}} = |A| = |A_s| \cdot |A_u| = 6. \quad (4.3)$$

La funzione di attivazione utilizzata in tutti i layer, eccetto l'ultimo, è la *tangente iperbolica* (\tanh) [46].

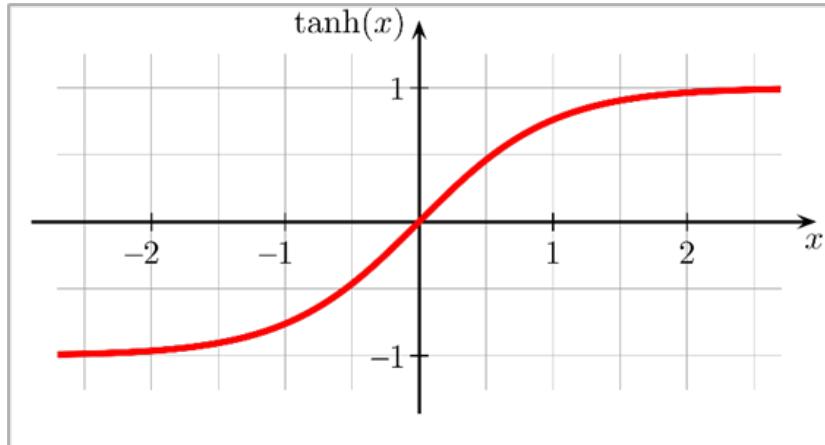


Figura 4.3: Grafico della funzione di attivazione \tanh [47].

Invece, la funzione di attivazione dell'output layer è la *softmax*, da cui poi dalla distribuzione ritornata si estraie il valore di a_i^t [48].

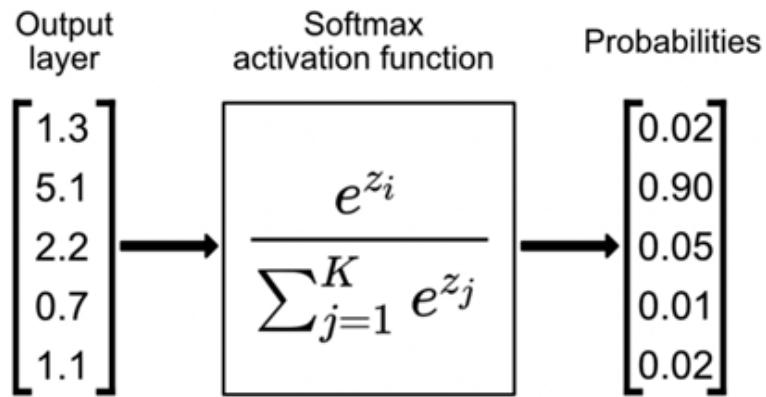


Figura 4.4: Illustrazione del funzionamento della softmax [48].

4.3 KPI

4.3.1 KPI di comunicazione

Per dare una valutazione alle prestazioni della policy appresa, è essenziale analizzare come essa si comporta al variare dei parametri di traffico di rete e delle condizioni di trasmissione. A tal fine, sono stati diversi KPI:

- **Pacchetti correttamente ricevuti:** Il numero di pacchetti correttamente ricevuti (N_{RX}) per ogni episodio di training/testing.
- **Goodput:** Il Goodput è definito come il numero di pacchetti ricevuti dalla Base Station per unità di tempo, senza considerare le ritrasmissioni:

$$G = \frac{N_{RX}}{T_{EP}}, \quad (4.4)$$

dove T_{EP} è il tempo necessario per completare il *task* di trasmissione. Questo parametro viene calcolato solo se il task viene completato prima di T_{EP} , che rappresenta la durata massima di un episodio.

- **Tempo in buffer:** Il tempo trascorso in buffer per ogni pacchetto di ciascun agente, viene calcolato come segue:

$$T_{BUFFER} = T_{ST} - T_G, \quad (4.5)$$

dove T_{ST} è lo step temporale in cui il pacchetto viene correttamente ricevuto dalla Base station e T_G è il momento in cui il pacchetto è stato generato.

- **Collisioni:** Il numero di collisioni che si è verificato durante un episodio di training/testing è calcolato considerando le collisioni per ciascun agente. Le collisioni si verificano quando più agenti tentano di inviare pacchetti nello stesso step temporale utilizzando l'unica risorsa disponibile.
- **Ritrasmissioni:** Il numero di volte in cui un pacchetto già ricevuto con successo dalla Base Station è stato ritrasmesso dall'agente. Questa metrica è di grande importanza in ambienti industriali, poiché ritrasmissioni inutili comportano un notevole spreco di energia.
- **Affidabilità dei pacchetti (Packet Reliability):**

$$R_P = \frac{\sum_i N_{RX_i}}{N_p \cdot N_{ep}}, \quad (4.6)$$

dove N_{RX_i} rappresenta i pacchetti correttamente ricevuti al i -esimo episodio, N_p è il numero totale di pacchetti da trasmettere e N_{ep} è il numero di simulazioni. Misura il tasso medio di successo della trasmissione dei pacchetti attraverso più simulazioni.

4.3.2 KPI di efficienza

Per confrontare l'efficienza tra i due modelli sono stati scelti i seguenti KPI:

- Tempo di inferenza: Misura in millisecondi (ms) il tempo necessario affinché il modello elabori l'input e restituisca l'output;
- Memoria occupata dal modello durante l'esecuzione;
- Memoria occupata dal modello sul disco.

```
1 # Warm-up
2 with torch.inference_mode():
3     for _ in range(1000):
4         model(input_tensor)
5
6 # Misurazione
7 times = []
8 start_time = time.perf_counter()
9 with torch.inference_mode():
10    for _ in range(4000):
11        model(input_tensor)
12 end_time = time.perf_counter()
13
14 avg_time = (end_time - start_time) * 1000 / 4000
15 print(f"Tempo medio di inferenza: {avg_time:.3f} ms")
```

Codice 4.2: Codice python - Misurazione del tempo di inferenza.

4.4 Applicazione della quantizzazione

Come già discusso nel capitolo precedente, l'utilizzo della quantizzazione rappresenta una soluzione efficace per ridurre il consumo di memoria e i tempi di inferenza, rendendo i modelli DNN più adatti all'esecuzione su IoTD. Per ottenere ciò, nel modello proposto sono state applicate tutte le tecniche discusse nel capitolo precedente, ovvero: La quantizzazione statica (Post-training), la quantizzazione dinamica (Post-training) e la quantization aware training;

Uno dei problemi principali della quantizzazione, già discusso nel capitolo 3, è che non tutte le operazioni sono supportate nel dominio quantizzato (Es: Softmax) quindi è importante modificare il proprio modello in modo tale da effettuare questo tipo di

operazioni dopo la dequantizzazione del valore in output. Per esempio, nel caso proposto la struttura del modello da quantizzare si è modificata come segue:

```

16 self.actor = nn.Sequential(
17     nn.Linear(state_dim, number_of_hidden_nodes),
18     nn.Tanh(),
19     nn.Linear(number_of_hidden_nodes, number_of_hidden_nodes),
20     nn.Tanh(),
21     nn.Linear(number_of_hidden_nodes, action_dim)
22 )

```

Codice 4.3: Codice python - Struttura del modello quantizzato.

In questo modo, è stato possibile eseguire nel modello quantizzato un'operazione di dequant tra l'uscita del layer sequenziale e la softmax senza errori.

```

23 def forward(self, x):
24     x = torch.quantize_per_tensor(x, self.scale, self.zero_point, torch.
25         quint8).unsqueeze(0)
26     x = self.actor(x)
27     x = torch.softmax(torch.dequantize(x), dim=-1)
28     dist = Categorical(x)
29     action = dist.sample()
30     action_logprob = dist.log_prob(action)
31     return action.detach(), action_logprob.detach()
32

```

Codice 4.1: Codice python - Forward del modello quantizzato.

4.4.1 Applicazione della quantizzazione dinamica

Per applicare la quantizzazione dinamica è stata utilizzata la funzione *quantize_dynamic* della libreria *PyTorch* [41]. Questa funzione restituisce il modello quantizzato e prende in ingresso diversi parametri, tra cui:

- Il modello float32 da quantizzare;
- L'argomento *Q_config_spec*, che può essere:
 - Un dizionario che mappa il nome o il tipo di sottomodulo alla configurazione di quantizzazione (qconfig) adottata. La configurazione specificata sarà

applicata a tutti i sottomoduli di un determinato modulo, a meno che non sia definita una configurazione specifica per i singoli sottomoduli.

- Un insieme contenente i tipi (es. `torch.nn.Linear`) o i nomi dei sottomoduli.

In questo caso, la quantizzazione dinamica sarà applicata solo a questi tipi o nomi. Questo campo presenta dei limiti, ovvero, non è possibile applicare la quantizzazione dinamica ai moduli convoluzionali;

- Il flag *Inplace*, se `True`, il modello originale viene modificato direttamente e sostituito con il modello quantizzato. Se `False` il modello quantizzato è una copia del modello originale, che non cambia. Se non specificato il valore predefinito è `False`;
- L'argomento *dtype*, che viene utilizzato per specificare la larghezza di bit; Per esempio, `qint8` o `quint8`; Il parametro opzionale *Mapping*, che consente di specificare una mappatura personalizzata tra i tipi di moduli originali e i corrispondenti tipi di moduli quantizzati.

Nell'esempio rappresentato nel codice a seguire, durante la quantizzazione verranno quantizzati solo i layer *Linear* tramite una rappresentazione `qint8`; inoltre, nella mappatura viene specificato con quali tipi di layer devono essere sostituiti.

```

1 model_q=torch.quantization.quantize_dynamic(
2     model,
3     q_config_spec={torch.nn.Linear},
4     dtype=torch.qint8,
5     mapping={Torch.nn.Linear: torch.nn.quantized.dynamic.Linear}
6 )

```

Codice 4.1: Codice python - Esempio di quantizzazione dinamica.

Invece, nei codici seguenti sono rappresentate le configurazioni adottate nel modello proposto e che verranno valutate nel prossimo capitolo.

```

1     model_final=torch.quantization.quantize_dynamic(
2         model_final,
3         q_config_spec={nn.Sequential},
4         dtype=torch.qint8
5     )

```

Codice 4.1: Codice python - Prima configurazione.

```

1 model_final=torch.quantization.quantize_dynamic(
2     model_final,
3     q_config_spec={nn.Sequential},
4     dtype=torch.float16
5 )

```

Codice 4.1: Codice python - Seconda configurazione.

4.4.2 Applicazione della quantizzazione statica

Per applicare la quantizzazione statica è stata utilizzata sempre la libreria PyTorch; Per prima cosa è stata definita le configurazioni da adottare al variare di M , raffigurata nei codici seguenti;

```

1 layer_config = M
2 match layer_config:
3     case 0:
4         qconfig = torch.quantization.get_default_qconfig(model_config)

```

Codice 4.1: Codice python - Configurazione quantizzazione statica

```

1     case 1:
2         qconfig = torch.quantization.QConfig(
3             activation=torch.quantization.MinMaxObserver(
4                 with_args(
5                     dtype=torch.quint8,
6                     reduce_range=True,
7                     quant_min=0,
8                     quant_max=127),
9             weight=torch.quantization.PerChannelMinMaxObserver(
10                with_args(
11                    dtype=torch.qint8,
12                    reduce_range=True,
13                    quant_min=0,
14                    quant_max=16)))

```

Codice 4.1: Codice python - Configurazione custom $M = 1$

```

1      case 3:
2          qconfig = torch.quantization.QConfig(
3              activation = torch.quantization.MinMaxObserver .
4                  with_args(
5                      dtype = torch.quint8 ,
6                      reduce_range = True ,
7                      quant_min = 0 ,
8                      quant_max = 127) ,
9              weight = torch.quantization.PerChannelMinMaxObserver .
10                 with_args(
11                     dtype = torch.qint8 ,
12                     reduce_range = True ,
13                     quant_min = 0 ,
14                     quant_max = 15))

```

Codice 4.1: Codice python - Configurazione custom $M = 2$

```

1      case 4:
2          qconfig = torch.quantization.QConfig(
3              activation = torch.quantization.MinMaxObserver .
4                  with_args(
5                      dtype = torch.quint8 ,
6                      reduce_range = True ,
7                      quant_min = 0 ,
8                      quant_max = 127) ,
9              weight = torch.quantization.PerChannelMinMaxObserver .
10                 with_args(
11                     dtype = torch.qint8 ,
12                     reduce_range = True ,
13                     quant_min = 0 ,
14                     quant_max = 127))

```

Codice 4.1: Codice python - Configurazione custom $M = 3, 4$

Il case θ è stato utilizzato per testare tre configurazioni di default:

- **Configurazione A:** Le attivazioni sono monitorate utilizzando un `HistogramObserver`, mentre, i pesi sono quantizzati per canale.
- **Configurazione B:** Le attivazioni sono monitorate con un `HistogramObserver` che applica una riduzione del range. Questo metodo migliora l'efficienza computazionale, poiché un range più ristretto consente una rappresentazione più compatta

dei dati, a fronte di una lieve perdita di precisione; mentre, i pesi sono quantizzati per canale.

- **Configurazione C:** Le attivazioni utilizzano un `HistogramObserver`. Mentre, i pesi sono quantizzati per tensore utilizzando un `MinMaxObserver`.

Mentre i casi 1,2,3,4, sono configurazioni personalizzate da usare in base al valore di M .

In base a quanto discusso nel capitolo 3, dopo la scelta della configurazione dovrebbe esserci il *fused step*, che in questo caso, non può essere implementato in quanto non è supportata la fusione tra un layer lineare e la funzione di attivazione tanh. Quindi si è passato direttamente all'utilizzo della funzione *prepare* della libreria PyTorch [49]; Questa funzione prepara il modello allo step successivo, ovvero la calibrazione, aggiungendo gli observer secondo il tipo di configurazione adottato.

La *prepare* restituisce il modello pronto alla calibrazione e prende in ingresso diversi parametri, come:

- Il modello da quantizzare;
- *Inplace*;
- *Allow_list*: Questo parametro opzionale contiene una lista che contiene i tipi di layer a cui applicare la quantizzazione, con questo si evita che altri tipi di layer non desiderati vengano quantizzati.
- *Observer_non_leaf_module_list*: Generalmente, gli osservatori vengono applicati solo ai moduli foglia, ovvero quelli che producono i risultati finali. Tuttavia, in alcuni casi, potrebbe essere necessario aggiungere osservatori anche ai moduli non foglia per raccogliere statistiche più complete durante la fase di calibrazione, migliorando così la precisione della quantizzazione. Questo campo è *opzionale*;
- *Prepare_custom_config_dict*: Questo parametro opzionale contiene un dizionario che permette di personalizzare la configurazione dei singoli layer/attivazioni.

Dopo la fase di preparazione del modello, è stata eseguita la sua calibrazione. La calibrazione è stata effettuata utilizzando un numero prestabilito di dati, generati tramite un procedimento che seleziona casualmente un valore all'interno di un intervallo definito dai range minimo e massimo degli input reali. La funzione responsabile della generazione di questi dati è la funzione *dummy_input*, che include un seed per garantire che

vengano restituiti sempre gli stessi input. In questo modo, impostando un range minimo, un range massimo e il numero di input da generare, tutte le configurazioni vengono calibrate con gli stessi dati, permettendo così un confronto più equilibrato.

```

1 def dummy_input(dim_input, num_points, min_range, max_range):
2     # Inserimento di un seed per valutare le performance sugli stessi
        input di calibrazione
3     random.seed(0)
4     # Genera num_points punti casuali in un range di valori definito dal
        input_min_range_vect e da input_max_range_vect
5     return [[random.randint(min_range[j], max_range[j]) for j in range(
        dim_input)] for _ in range(num_points)]
6 def calibration(model):
7     input_min_range_vect = [0] * dim_states_ag
8     input_max_range_vect = [P_tr]
9     input_vect = [2, 2, 1, P_tr]
10    for l in list(range(N_ag)):
11        input_max_range_vect = input_max_range_vect + input_vect
12    dummy_states = dummy_input(dim_states_ag, number_dummy_points,
        input_min_range_vect, input_max_range_vect)
13    for input_cab in dummy_states:
14        model.forward(torch.tensor(input_cab).float())

```

Codice 4.1: Codice python della calibrazione

Infine, è stata eseguita la conversione del modello tramite la funzione *convert* della libreria PyTorch che ritorna il modello quantizzato [50]. Essa ha come parametri in ingresso:

- *Modello calibrato*;
- *Mapping*;
- *Inplace*;
- *Use_precomputed_fake_quant*: Un flag booleano che, se impostato su `True`, abilita l'uso di moduli *fake quant* precalcolati.

Nel capitolo successivo verranno valutate diverse configurazioni adottate per analizzare come queste influenzino le prestazioni del modello. Le configurazioni saranno ottenute modificando parametri specifici, come la configurazione di quantizzazione adottata, il range degli osservatori e il numero di input per la calibrazione.

Inoltre, verrà valutato l'impatto del numero di input di calibrazione attraverso il calcolo della divergenza di *Kullback-Leibler* (KL) per osservare eventuali effetti sulle prestazioni. La divergenza KL rappresenta una misura che, più è bassa, maggiore è la somiglianza tra le distribuzioni [ref].

La formula per calcolare tale divergenza tra due distribuzioni P e Q è:

$$D_{KL}(P \parallel Q) = \sum_i P(i) \log_2 \left(\frac{P(i)}{Q(i)} \right) = \sum_i P(i) (\log_2 P(i) - \log_2 Q(i)) \quad (4.7)$$

```
1 res_original = collectResult(dataset, model_original)
2 res_quantized = collectResult(dataset, model_q)
3 kl_res = []
4 for dis_original, dis_quantized in zip(res_original, res_quantized):
5     # Se ci sono valori uguali a 0 il log non si può fare quindi possiamo
      aggiungere un piccolo valore
6     # Parte commentata in quanto non ho riscontrato questo problema
7     # epsilon = 1e-12
8     # dis_original = dis_original + epsilon
9     # dis_quantized = dis_quantized + epsilon
10    # FORMULA KL
11    kl = (dis_original * (dis_original.log() - dis_quantized.log())).sum()
12    kl_res.append(kl)
```

Codice 4.1: Codice python divergenza KL

La divergenza KL sarà poi analizzata attraverso due tipi di grafici:

- **Grafici delle Distribuzioni**, che mostrano come si distribuiscono i valori della divergenza KL;
- **Whisker Plot**, che evidenziano i valori della divergenza KL, mostrando *media* e *mediana*. La media è il valore ottenuto sommando tutti i dati e dividendo per il loro numero totale, mentre la mediana rappresenta il valore centrale dei dati ordinati. La media è sensibile ai valori estremi (outlier), mentre la mediana rappresenta meglio il valore tipico quando vi sono dati anomali [51].

Nel contesto di questa analisi, il processo inizia con la creazione di un dataset che raccoglie tutte i possibili input del modello tramite la funzione *createDataset()*.

```

1 def createDataset():
2     val_max = [P_tr]
3     input_vect = [2, 2, 1, P_tr]
4     for l in list(range(N_ag)):
5         val_max = val_max + input_vect
6     ranges = []
7     for max_val in val_max:
8         # Per ogni posizione del vettore, prende il valore, e crea una
9         # sequenza di valori che va da 0 a quel valore - 1
10        # per esempio se 2, crea una sequenza di valori che comprende 0 e
11        # 1; max_val + 1 serve per includere il valore massimo
12        # Queste sequenze vengono salvate nella stessa posizione del
13        # valore che stiamo esaminando
14        ranges.append(range(max_val + 1))
15
16
17    # La funzione product ci permette di ottenere tutte le possibili
18    # combinazioni tra le sequenze trovate, mantenendo l'ordine delle
19    # posizioni
20
21    # ovvero, in base a come sono state inserite nel vettore
22    combinations = product(*ranges)
23
24
25    # Converte il risultato in una lista per avere tutte le combinazioni
26    # in dataset
27
28    return list(combinations)

```

Codice 4.2: Codice python creazione dataset

Il dataset è stato poi passato sia al modello originale che al modello quantizzato, in modo da ottenere le rispettive distribuzioni e calcolare la divergenza di Kullback-Leibler (KL).

Sono state testate diverse configurazioni, ciascuna determinata da:

- Un valore associato alla memoria M ;
- Il tipo di configurazione di quantizzazione adottato; Che in questo caso sono tutte quelle citate nel caso θ .
- Gli input utilizzati per la calibrazione; questi sono scelti randomicamente e il loro numero può variare tra il 5%, 10%, 25%, 50%, 75% e 100% della lunghezza totale del dataset.

Questo processo aumenta di complessità computazionale all'aumentare della memoria, poiché aumenta il numero di elementi del dataset; quindi, si è escluso dal processo il valore $M = 4$, in quanto il processo era troppo dispendioso. Sulla base dei risultati ottenuti, per ogni coppia (M , configurazione di quantizzazione) sono state selezionate due configurazioni ottimali: una configurazione che riduce la media della divergenza KL ed una configurazione che ne riduce la mediana; queste poi sono state confrontate con il modello originale.

4.4.3 Applicazione della Training Aware Quantization

Per applicare la QAT al modello, è stata utilizzata la libreria PyTorch. Il primo passo consiste nella scelta della configurazione da adottare. In una prima fase è stata scelta la configurazione A, in una seconda fase, invece, è stata utilizzata una configurazione personalizzata, illustrata di seguito.

```
1 custom_qconfig = torch.ao.quantization.QConfig(
2     activation=torch.ao.quantization.observer.MovingAverageMinMaxObserver
3         .with_args(
4             dtype=torch.quint8,
5             qscheme=torch.per_tensor_affine,
6             reduce_range=True,
7             quant_min=0, # Valore minimo quantizzato
8             quant_max=127 # Valore massimo quantizzato
9         ),
10    weight=torch.ao.quantization.observer.MinMaxObserver.with_args(
11        dtype=torch.qint8,
12        qscheme=torch.per_tensor_symmetric,
13        reduce_range=False,
14        quant_min=-127, # Valore minimo quantizzato
15        quant_max=127 # Valore massimo quantizzato
16    )
17 )  
18 model_policy.qconfig = custom_qconfig
```

Codice 4.1: Codice python della configurazione QAT

Dopo aver scelto il tipo di configurazione adottare, si usa la funzione *prepare_qat* per aggiungere al modello gli osservatori e i fake quant. Questa funzione ritorna il modello pronto per essere addestrato in maniera consapevole della quantizzazione, e i suoi parametri in ingresso sono:

- Modello da quantizzare;
- *Inplace*;
- *Mapping*.

```

1 model_policy.qconfig = custom_qconfig
2 model_policy.train()
3 torch.quantization.prepare_qat(model_policy, inplace=True)
4
5 self.policy = model_policy
6 self.optimizer = torch.optim.Adam([
7     {'params': self.policy.actor.parameters(), 'lr': lr_actor},
8     {'params': self.policy.critic.parameters(), 'lr': lr_critic}
9 ])
10 self.policy_old = model_policy
11 self.policy_old.load_state_dict(self.policy.state_dict())
12 self.MseLoss = nn.MSELoss()
```

Codice 4.1: Codice python rappresentante la prepare e la policy adottata

Il modello ottenuto dalla *prepare_qat* va assegnato sia all'attore che al critico all'interno dell'algoritmo MAPPO, poi si può procedere con il training del modello; Una volta finito il training si passa alla fase di conversione del modello applicando la funzione *convert*, che rimuove i fake quant e inserisce le operazioni quantizzate.

```

1 # Salva lo stato del modello in un file
2 torch.quantization.convert(ppo_agents[0].policy, inplace=True)
3 ppo_agents[0].policy.eval()
4 save_prefix = 'models\state_QAT_dict_model_collision_reduced'
5 save_path = '{}_memory_{}_P_{}_ag_{}_exp_{}_abs_{}_node_{}_hid_{}_IDsim_{}_'.format(
6     save_prefix, N, P, n_agents, exp, 0, number_of_hidden_nodes,
    number_of_hidden_layers, ID_sim)
7 torch.save(ppo_agents[0].policy.state_dict(), save_path) # Salva lo
    stato della policy dell'agente PPo
```

Codice 4.1: Codice python rappresentante la convert e il salvataggio del modello addestrato

Nel capitolo successivo, le diverse configurazioni discusse verranno valutate per osservare come queste influenzano le prestazioni del modello. Poiché lo step finale della QAT riguarda il fine-tuning, le configurazioni considerate derivano anche dalla variazione di specifici iperparametri di training. Per la configurazione A, tutti gli iperparametri sono rimasti uguali a eccezione dei learning rate, in quanto è buona norma abbassarli in fase di finetuning. Nella configurazione Custom, invece, sono stati modificati diversi parametri con lo scopo di ottenere un modello migliore. I dettagli relativi a tali iperparametri sono illustrati nelle tabelle seguenti.

Parametro	Simbolo	Valore
Numero di simulazioni	N_{sim}	1000
Learning rate	L_r	$3 * 10^{-5}$
Learning rate crtic	L_c	10^{-4}

Tabella 4.2: Parametri fine-tuning - Configurazione A - $M = \{1, 2, 3, 4\}$.

Parametro	Simbolo	Valore
Numero di simulazioni	N_{sim}	1001
Discount factor	γ	0.96
Learning rate	L_r	$2 * 10^{-5}$
Learning rate crtic	L_c	10^{-4}
Clipping value	ϵ	0.3

Tabella 4.3: Parametri fine-tuning - Custom - $M = \{1, 2\}$.

Parametro	Simbolo	Valore
Numero di simulazioni	N_{sim}	1001
Discount factor	γ	0.99
Learning rate	L_r	$2 * 10^{-5}$
Learning rate crtic	L_c	$2 * 10^{-4}$
Clipping value	ϵ	0.27

Tabella 4.4: Parametri fine-tuning - Custom - $M = 3$.

Parametro	Simbolo	Valore
Numero di simulazioni	N_{sim}	1001
Discount factor	γ	0.96
Learning rate actor	L_r	$2 * 10^{-5}$
Learning rate critic	L_c	10^{-4}
Clipping value	ϵ	0.3

Tabella 4.5: Parametri fine-tuning - Custom - $M = 4$.

Capitolo 5

Risultati ottenuti

In questo capitolo verranno confrontati i KPI (illustrati in 4.3.1 e in 4.3.2) dei modelli quantizzati con quelli dei modelli originali, al fine di valutare l'efficacia della quantizzazione. Per garantire una valutazione accurata delle prestazioni, le simulazioni sono state condotte variando diversi *seed*, riducendo così il rischio che i risultati fossero influenzati da casi particolari legati a un singolo seed. I dati raccolti sono stati mediati, offrendo una visione più stabile e affidabile delle performance complessive dei modelli.

Parametro	Simbolo	Valore
Numero di dispositivi IoT	N	2
Numero di PDU da trasmettere	P	2
Durata massima dell'episodio	t_{max}	300
Lunghezza della memoria	M	{1, 2, 3, 4}
Tasso di errore sul blocco di trasmissione	BLER	0.01
Numero di simulazioni	N_{sim}	1000
Condizione di buffer iniziale	<i>case</i>	Full buffer
Seeds considerati	SEEDS	100

Tabella 5.1: Parametri della simulazione - quantizzazione dinamica.

5.1 Risultati - Dynamic quantization

Per quanto riguarda la quantizzazione dinamica, come accennato nel capitolo precedente, nel paragrafo 4.4.1, sono state adottate due configurazioni principali. Nelle tabelle seguenti sono riportati i risultati mediati, ottenuti con le diverse configurazioni della quantizzazione dinamica, confrontati con i risultati del modello originale.

KPI	Int8	Float16	Original
Goodput	0,4231	0,4238	0,4244
Time slots	15,5364	15,9588	15,8798
Packet	3,9808	3,9792	3,9795
Collision	0,6024	0,6057	0,6034
Already transm. packets	0,0126	0,0116	0,0113

Tabella 5.2: Risultati - Quantizzazione dinamica $M = 1$

KPI	Int8	Float16	Original
Goodput	0,4782	0,4761	0,4751
Time slots	13,8639	13,0698	13,2681
Packet	3,9819	3,9848	3,9842
Collision	0,5600	0,5683	0,5729
Already transm.packets	0,0674	0,0727	0,0739

Tabella 5.3: Risultati - Quantizzazione dinamica $M = 2$

KPI	Int8	Float16	Original
Goodput	0,4370	0,4371	0,4370
Time slots	21,6376	21,4247	21,5647
Packet	3,9555	3,9563	3,9559
Collision	0,5892	0,5907	0,5911
Already transm. packets	0,1228	0,1018	0,1013

Tabella 5.4: Risultati - Quantizzazione dinamica $M = 3$

KPI	Int8	Float16	Original
Goodput	0,4222	0,4210	0,4208
Time slots	23,1216	24,1642	23,9845
Packet	3,9517	3,9480	3,9848
Collision	0,4453	0,4566	0,4574
Already transm. packets	0,3718	0,3703	0,3678

Tabella 5.5: Risultati - Quantizzazione dinamica $M = 4$

Memory (M)	Int8	Float16	Original	Original-Only actor
$M = 1$	0,0121	0,0257	0,0430	0,0228
$M = 2$	0,0123	0,0267	0,0450	0,0238
$M = 3$	0,0126	0,0276	0,0469	0,0248
$M = 4$	0,0128	0,0286	0,0489	0,0257

Tabella 5.6: Memory usage on disk (Mb) - Quantizzazione dinamica

Memory (M)	Int8	Float16	Original	Original-Only actor
$M = 1$	0,0047	0,0094	0,0364	0,0188
$M = 2$	0,0049	0,0098	0,0383	0,0197
$M = 3$	0,0051	0,0103	0,0403	0,0207
$M = 4$	0,0054	0,0108	0,0422	0,0217

Tabella 5.7: Memory usage while running (Mb) - Quantizzazione dinamica

KPI	Int8	Float16	Original
Inference Time	0,0045	0,0043	0,0049

Tabella 5.8: Risultati Quantizzazione Dinamica - Tempi medi di inferenza a confronto.

Memoria Per quanto concerne la memoria, i modelli quantizzati vengono confrontati con il modello senza critico (only actor), poiché il critico non è necessario durante l'inferenza e occupa memoria inutilmente. In fase di esecuzione, il modello quantizzato con interi a 8 bit riduce l'occupazione di memoria del 75% rispetto al modello originale, mentre il modello quantizzato con 16 bit a virgola mobile occupa circa la metà della memoria del modello originale. Per quanto riguarda la memoria occupata su disco, il modello **Int8** riduce lo spazio richiesto del 50%, mentre il modello **Float16** supera leggermente le dimensioni del modello originale. Questo incremento di memoria è dovuto al fatto che i modelli quantizzati memorizzano nel file metadati aggiuntivi relativi alla quantizzazione.

Prestazioni Per quanto riguarda le prestazioni, entrambi i modelli quantizzati mostrano performance simili a quelle del modello originale. Come previsto, il modello **Float16** si avvicina maggiormente al comportamento del modello originale. Tuttavia, il modello **Int8** non si discosta di molto e, in alcuni contesti (ad esempio con $M = 4$ o $M = 2$), riesce persino a superarne di poco le prestazioni. Per quanto concerne il

tempo di inferenza, i modelli quantizzati sono più veloci di circa l'8% rispetto a quello originale; il modello **Float16** risulta leggermente più veloce rispetto a quello in **Int8**.

In conclusione, si può affermare che in questo caso la quantizzazione a 8 bit (**Int8**) è la soluzione più equilibrata, in quanto riduce drasticamente la memoria richiesta pur mantenendo prestazioni quasi identiche al modello originale con un leggero vantaggio in termini di tempo di inferenza.

5.2 Risultati - Static quantization

Per la quantizzazione statica, è stata innanzitutto calcolata la divergenza di Kullback-Leibler (KL) tra le distribuzioni di probabilità ottenute dal modello quantizzato e dal modello originale, seguendo il procedimento descritto in 4.4.2, i cui grafici sono riportati in 5.2.1 e i risultati in 5.2.2. Successivamente, sono state testate le configurazioni selezionate e confrontate con il modello originale, al fine di determinare il numero ideale di input di calibrazione e identificare la configurazione predefinita con le migliori prestazioni, per poi eseguire con tale configurazione delle simulazioni, i cui risultati sono riportati in 5.2.3. In seguito, sono state condotte ulteriori simulazioni utilizzando delle configurazioni personalizzate i cui risultati sono esposti nel paragrafo 5.2.4.

5.2.1 Risultati della divergenza di Kullback-Leibler

Distribuzione delle frequenze con la configurazione A e $M=1$

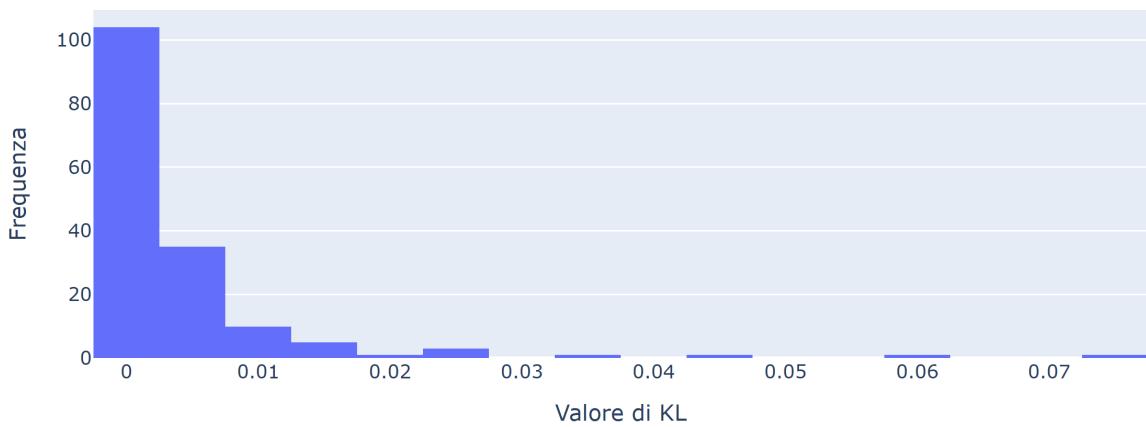


Figura 5.1: Grafici distribuzione |configurazione A, $M = 1$, input = 8|

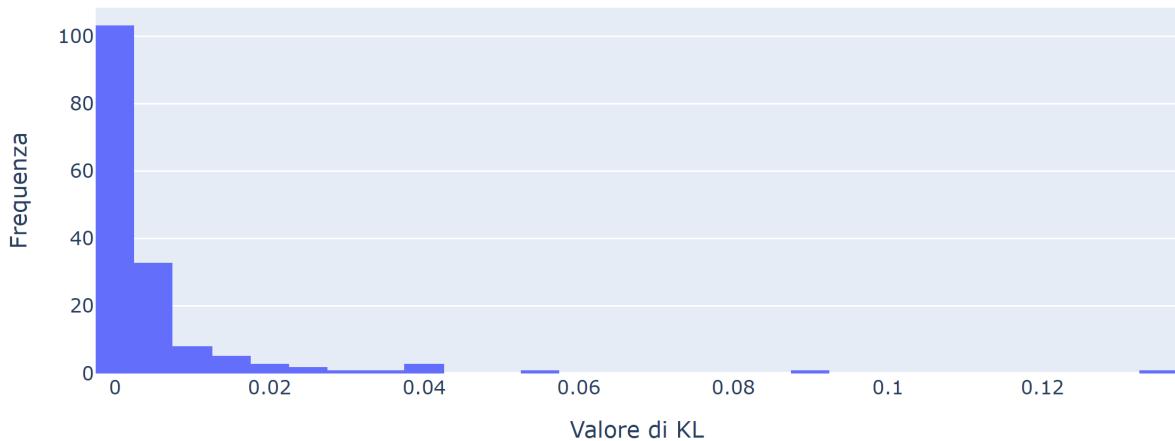


Figura 5.2: Grafici distribuzione |configurazione A, $M = 1$, input = 16|

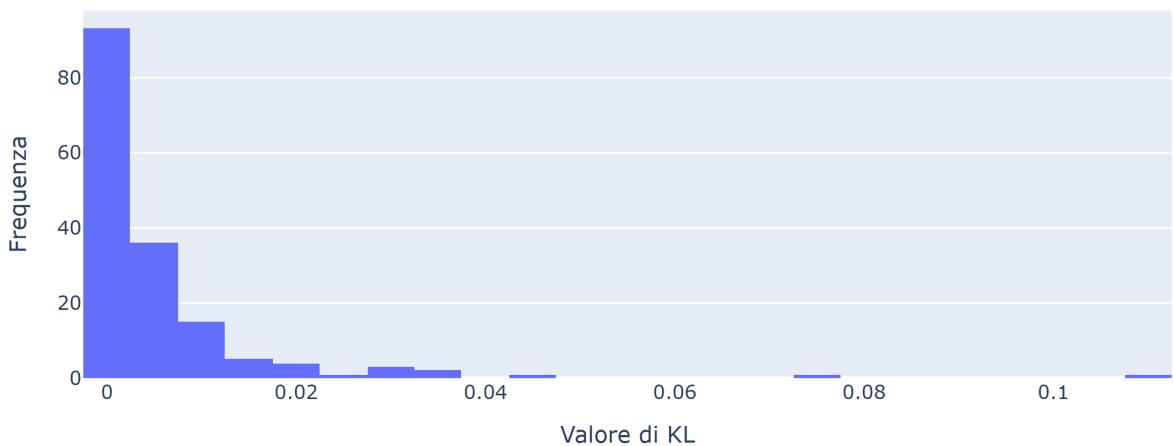


Figura 5.3: Grafici distribuzione |configurazione A, $M = 1$, input = 40|

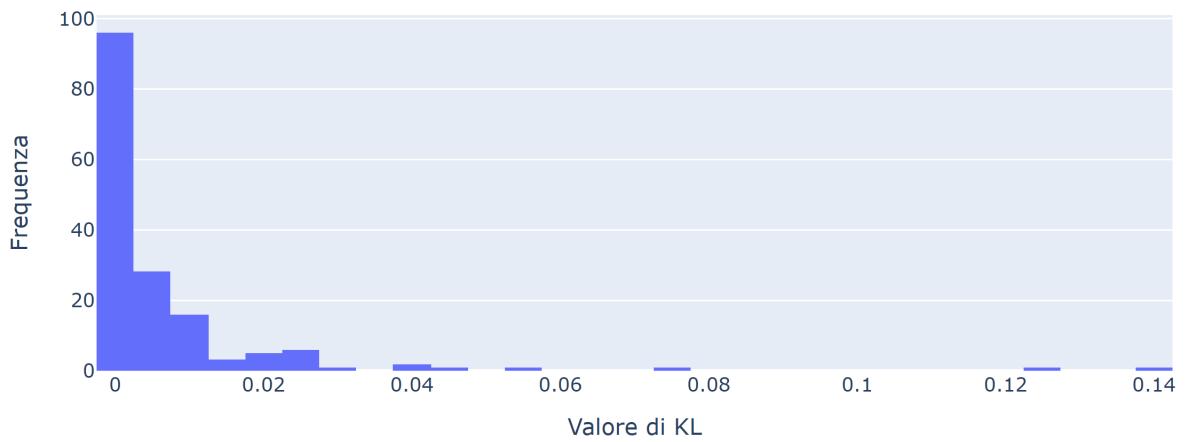


Figura 5.4: Grafici distribuzione |configurazione A, $M = 1$, input = 81|

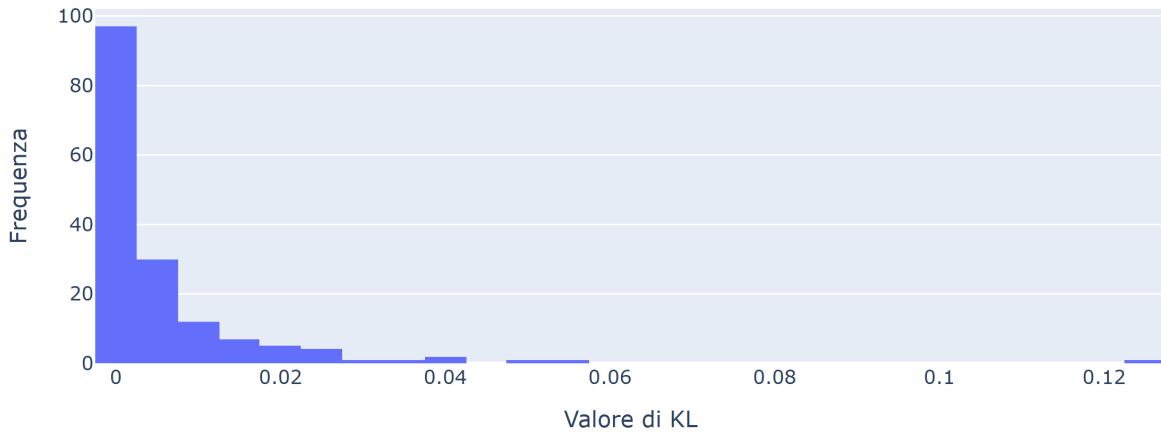


Figura 5.5: Grafici distribuzione |configurazione A, $M = 1$, input = 122|

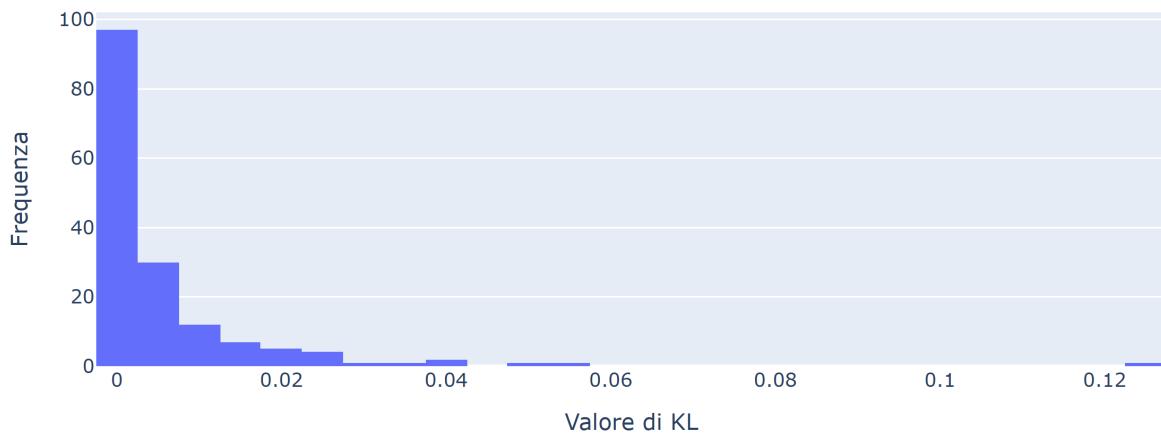


Figura 5.6: Grafici distribuzione |configurazione A, $M = 1$, input = 162|

Whisker Plot con la configurazione A e $M=1$

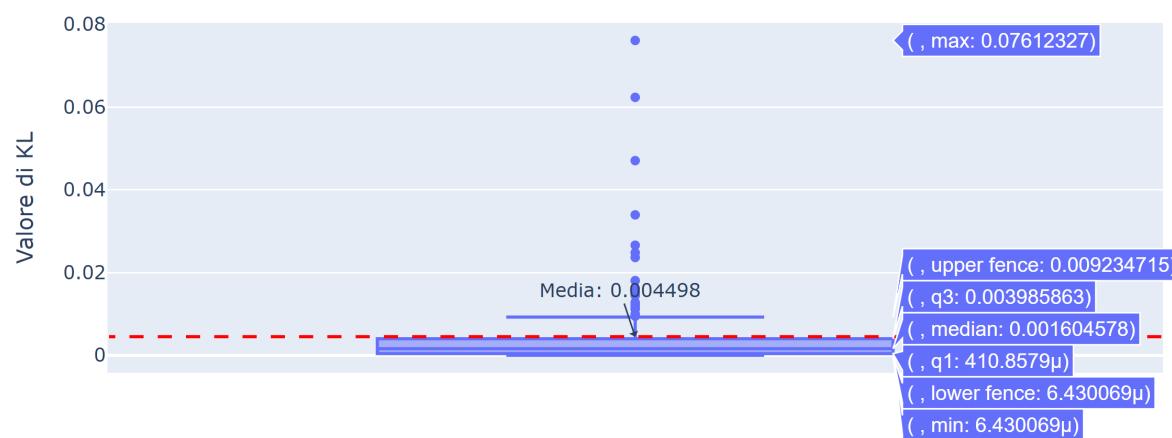


Figura 5.7: Whisker plot |configurazione A, $M = 1$, input = 8|

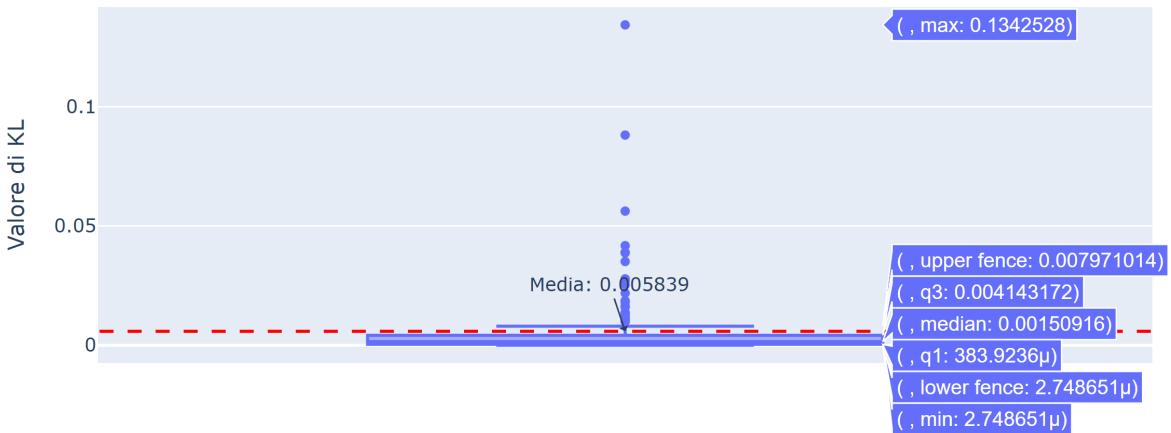


Figura 5.8: Whisker plot |configurazione A, $M = 1$, input = 16|

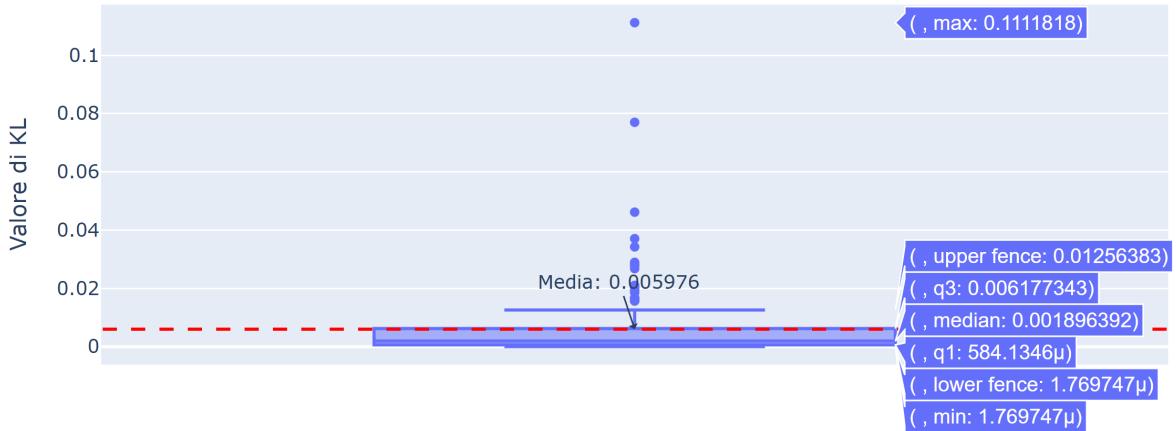


Figura 5.9: Whisker plot |configurazione A, $M = 1$, input = 40|

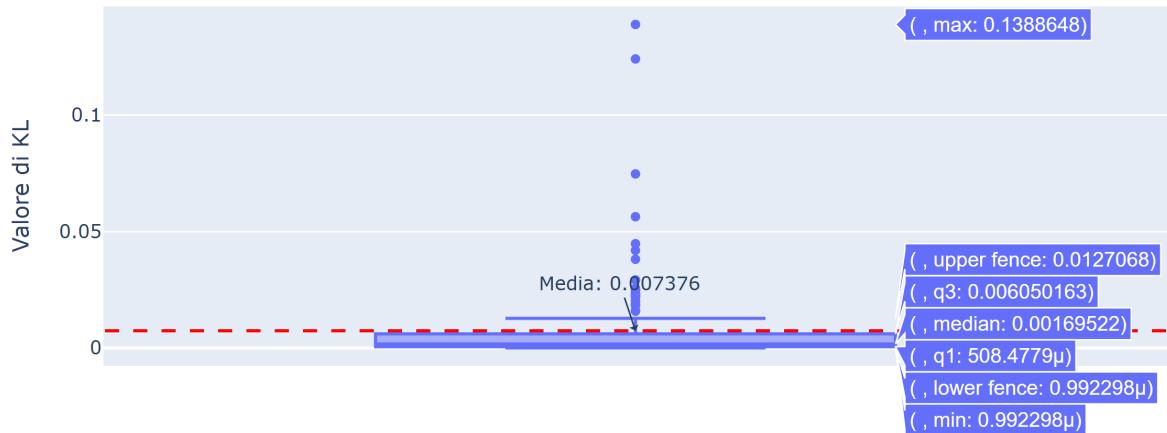


Figura 5.10: Whisker plot |configurazione A, $M = 1$, input = 81|

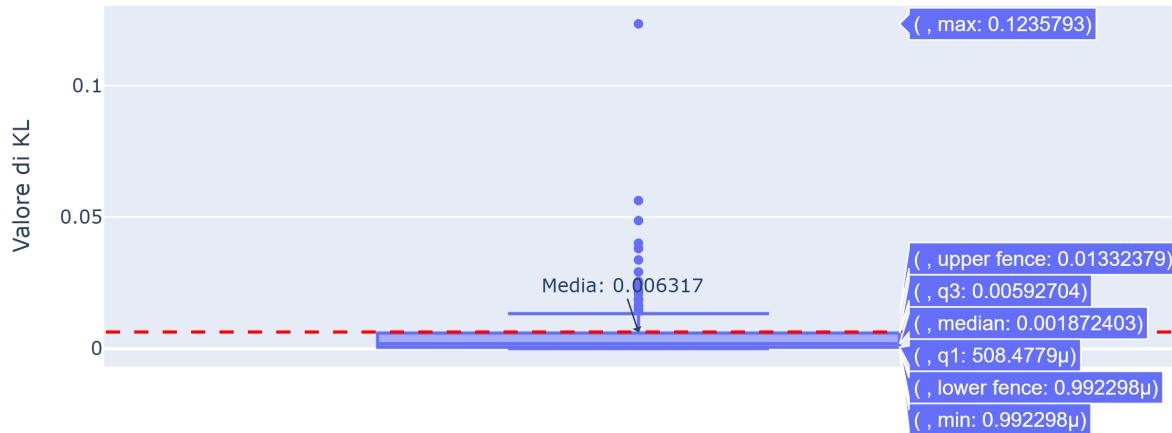


Figura 5.11: Whisker plot |configurazione A, $M = 1$, input = 122|

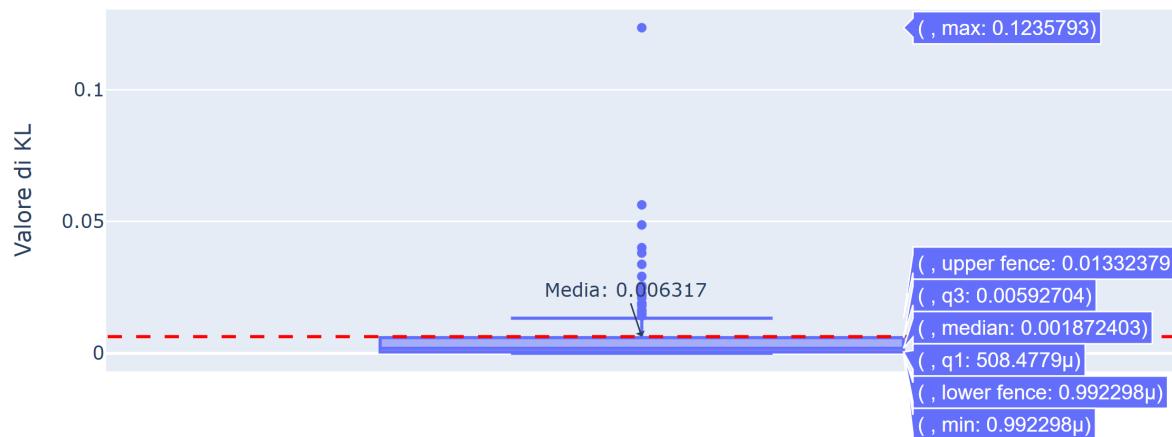


Figura 5.12: Whisker plot |configurazione A, $M = 1$, input = 162|

Distribuzione delle frequenze con la configurazione B e $M = 1$

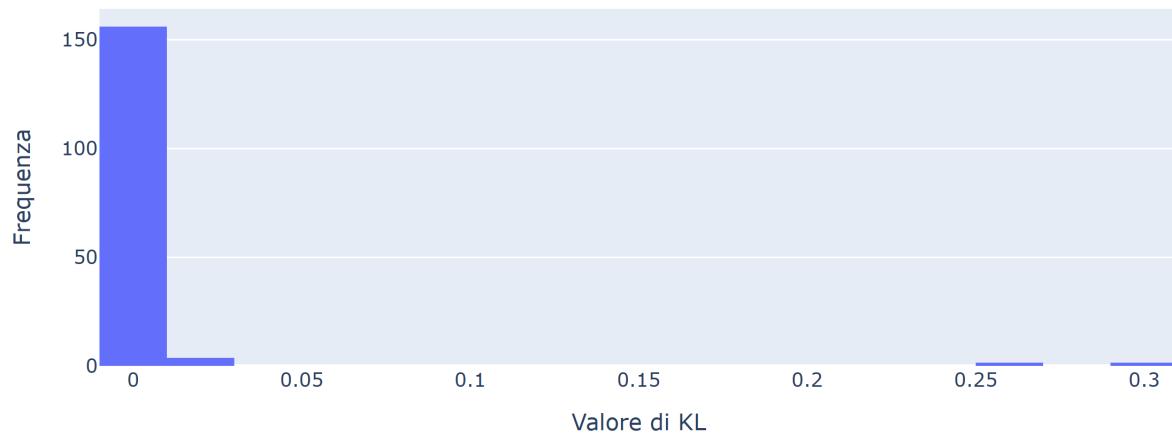


Figura 5.13: Grafici distribuzione |configurazione B, $M = 1$, input = 8|



Figura 5.14: Grafici distribuzione |configurazione B, $M = 1$, input = 16|

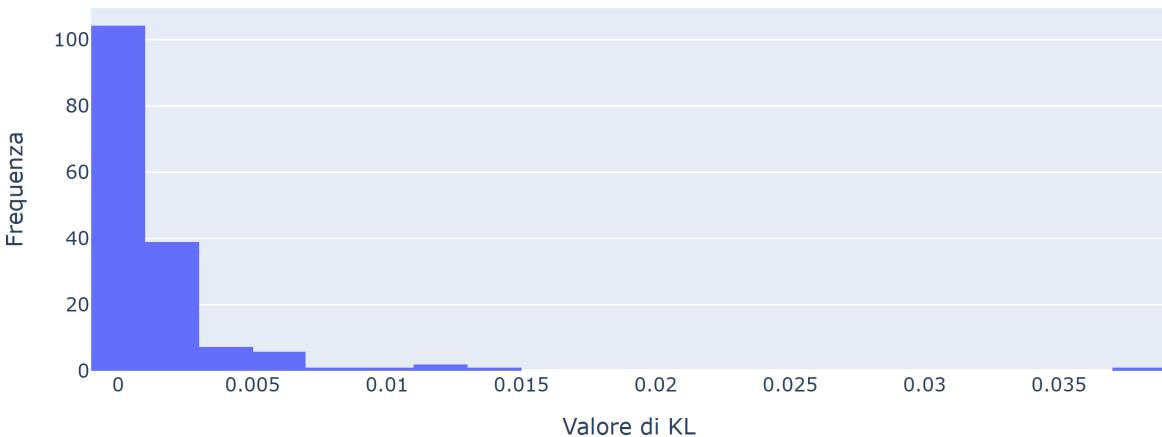


Figura 5.15: Grafici distribuzione |configurazione B, $M = 1$, input = 40|

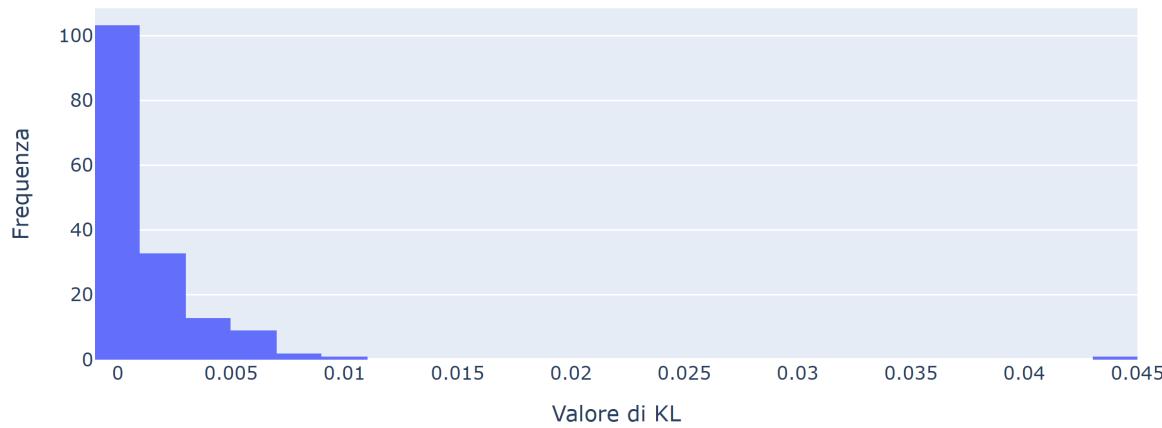


Figura 5.16: Grafici distribuzione |configurazione B, $M = 1$, input = 81|

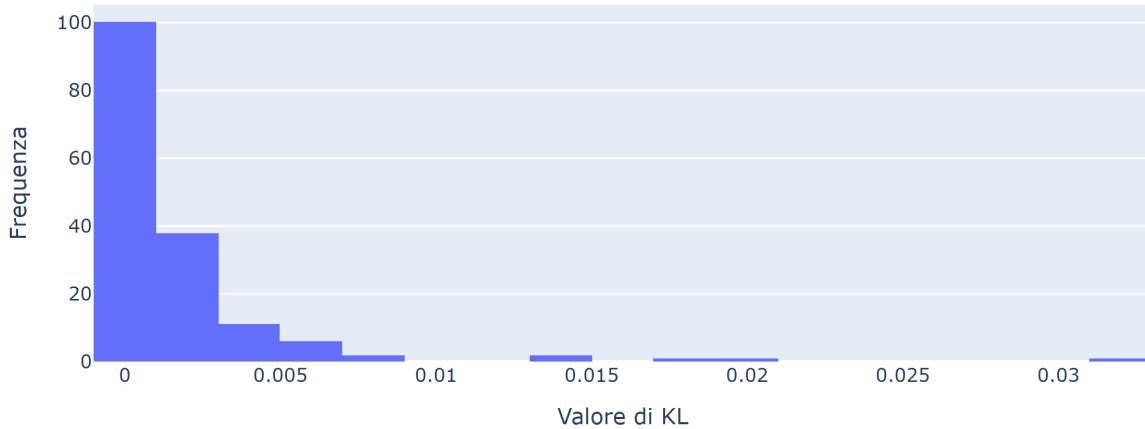


Figura 5.17: Grafici distribuzione |configurazione B, $M = 1$, input = 122|

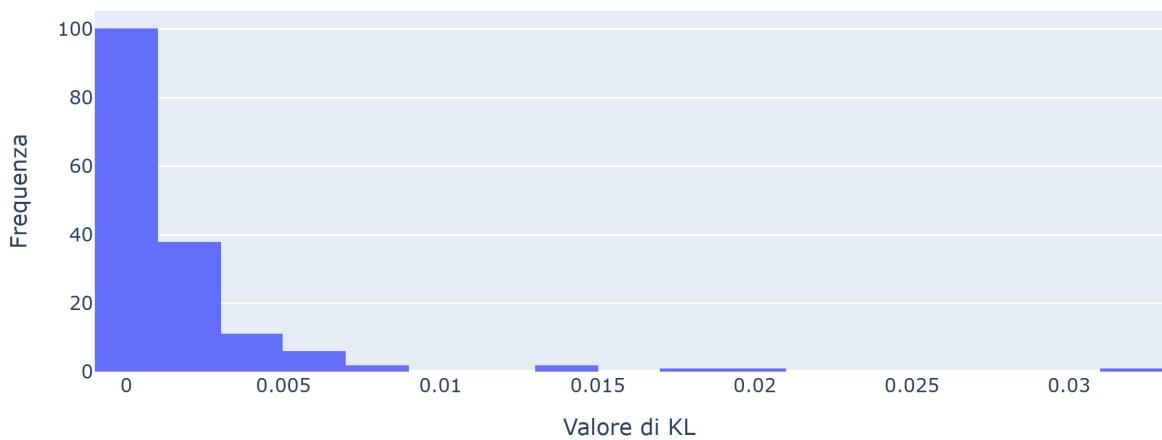


Figura 5.18: Grafici distribuzione |configurazione B, $M = 1$, input = 162|

Whisker Plot con la configurazione B e $M = 1$

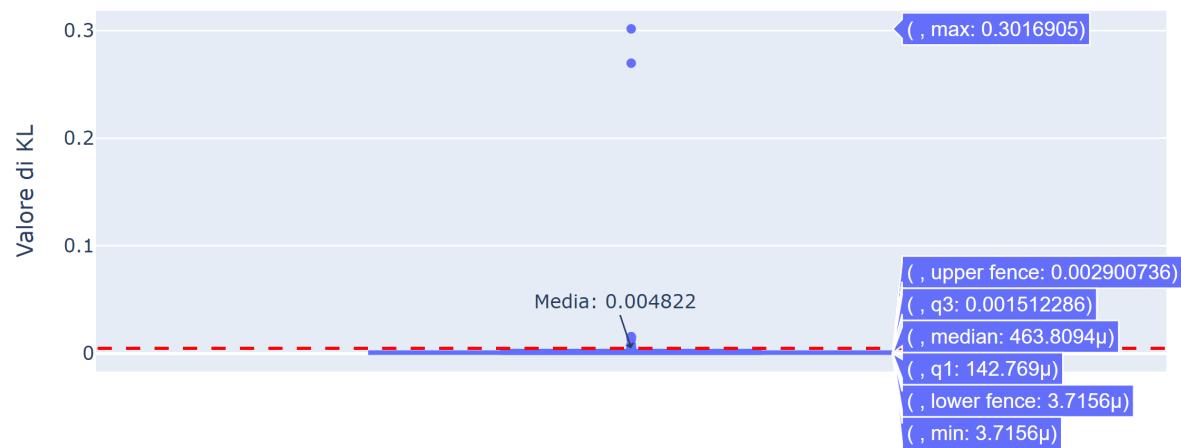


Figura 5.19: Whisker plot |configurazione B, $M = 1$, input = 8|

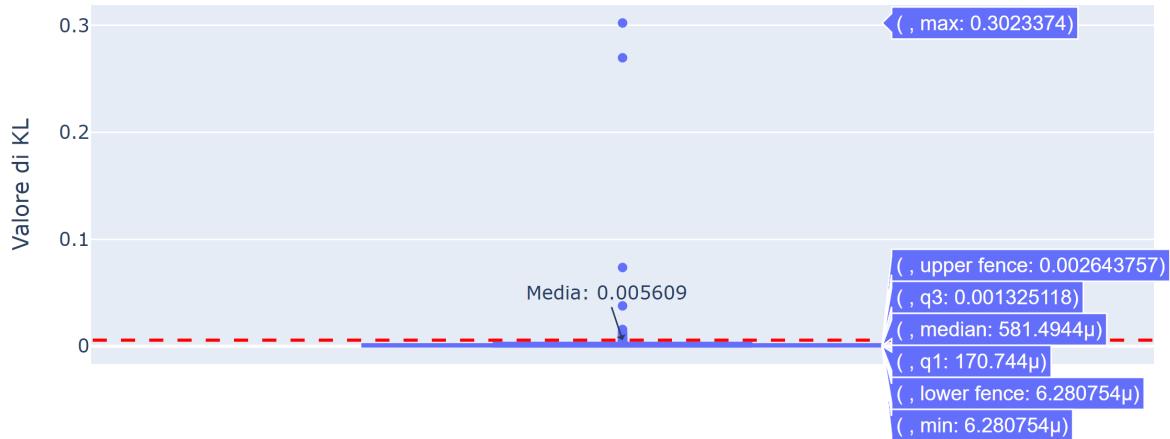


Figura 5.20: Whisker plot |configurazione B, $M = 1$, input = 16|

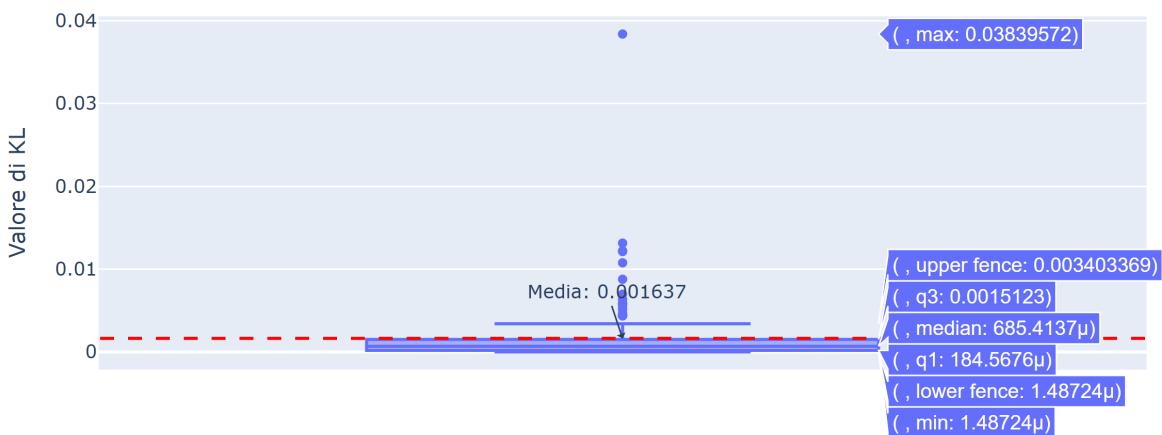


Figura 5.21: Whisker plot |configurazione B, $M = 1$, input = 40|

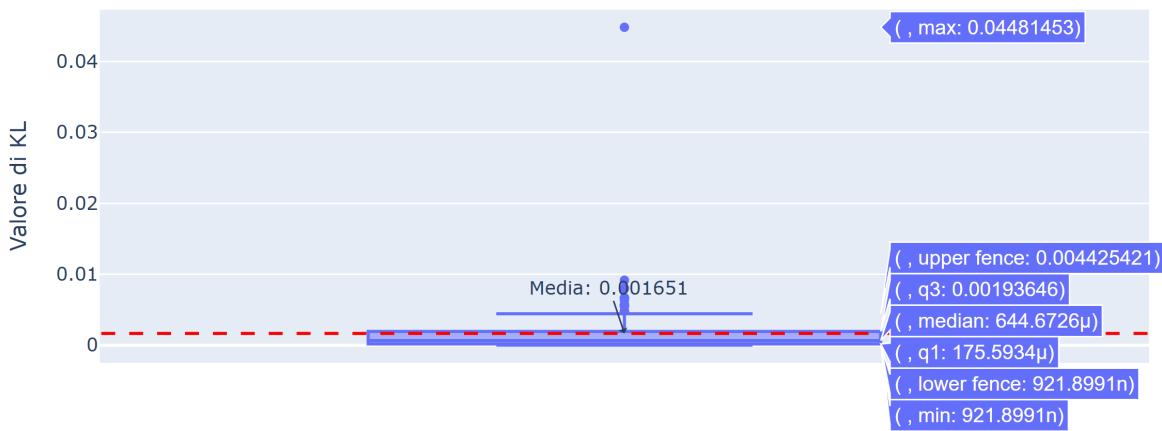


Figura 5.22: Whisker plot |configurazione B, $M = 1$, input = 81|

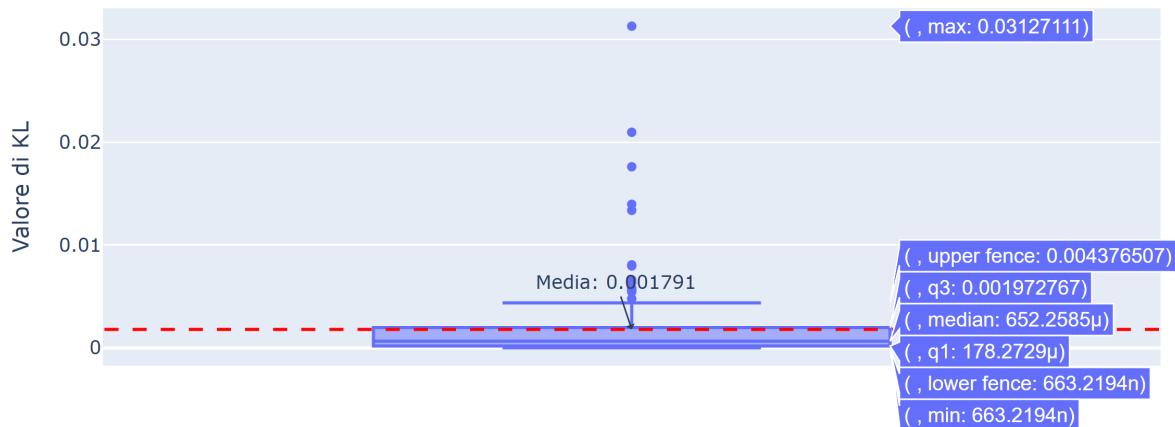


Figura 5.23: Whisker plot |configurazione B, $M = 1$, input = 122|

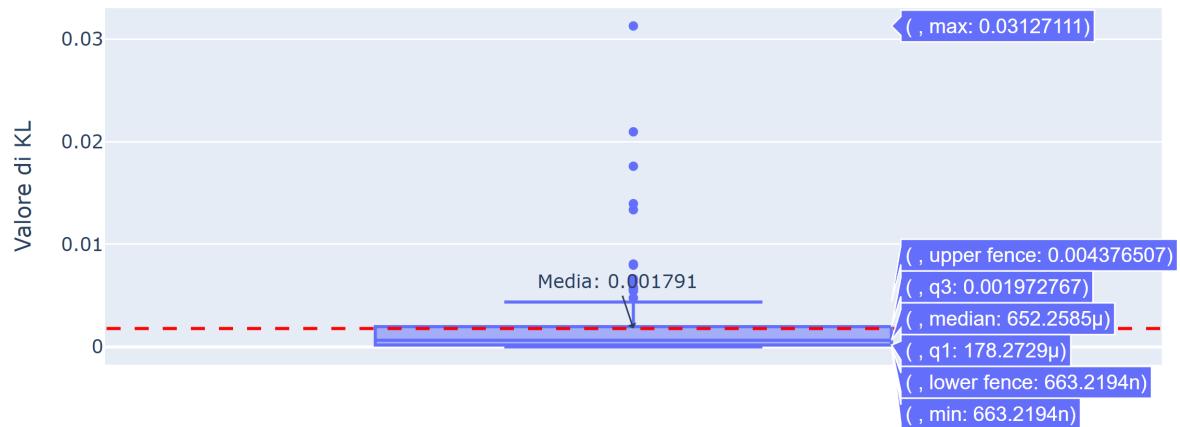


Figura 5.24: Whisker plot |configurazione B, $M = 1$, input = 162|

Distribuzione delle frequenze con la configurazione C e $M = 1$

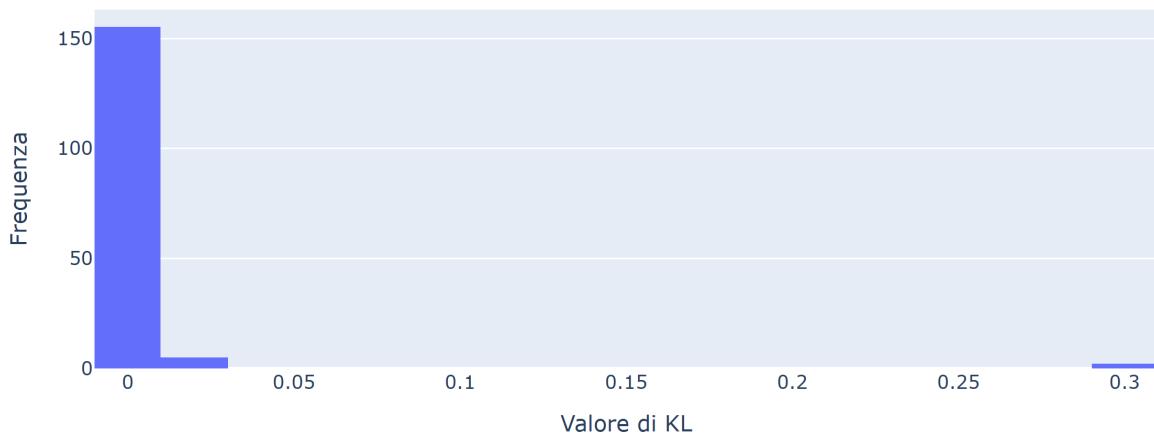


Figura 5.25: Grafici distribuzione |configurazione C, $M = 1$, input = 8|

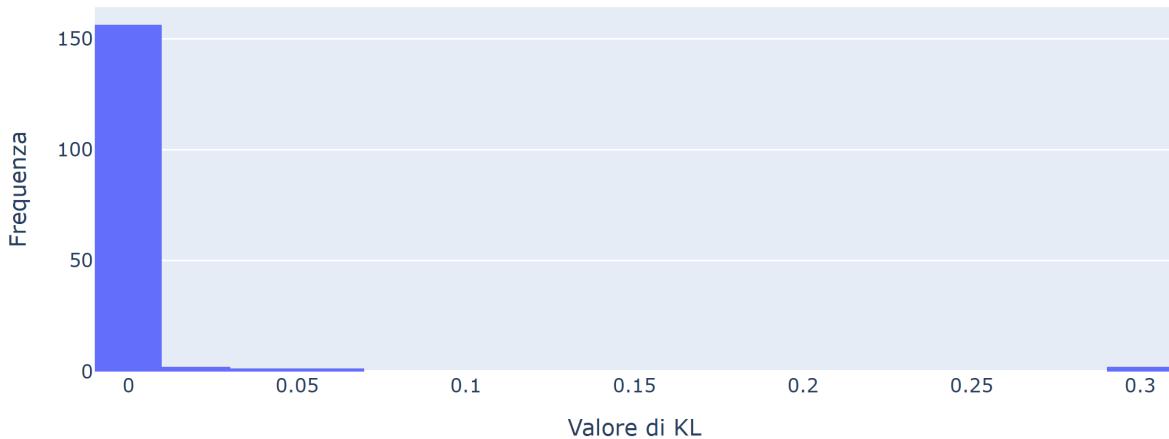


Figura 5.26: Grafici distribuzione |configurazione C, $M = 1$, input = 16|

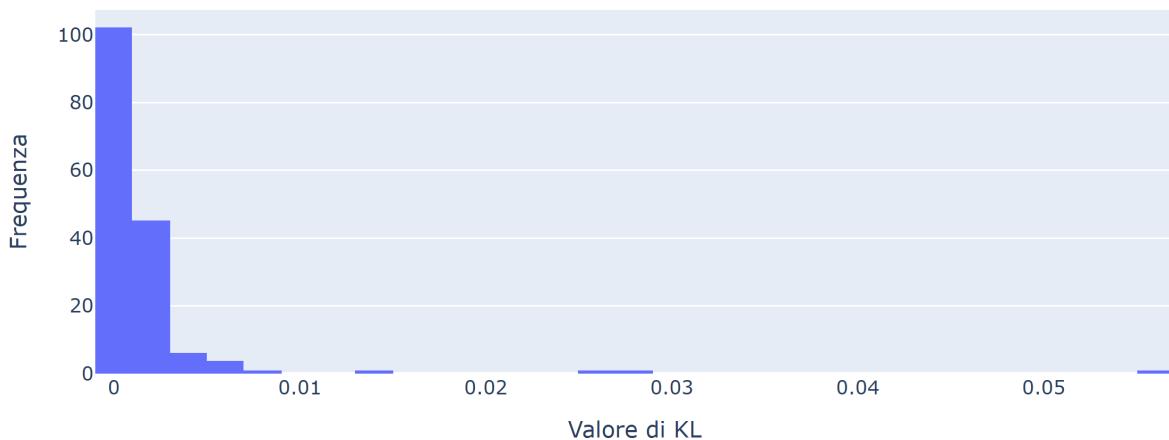


Figura 5.27: Grafici distribuzione |configurazione C, $M = 1$, input = 40|

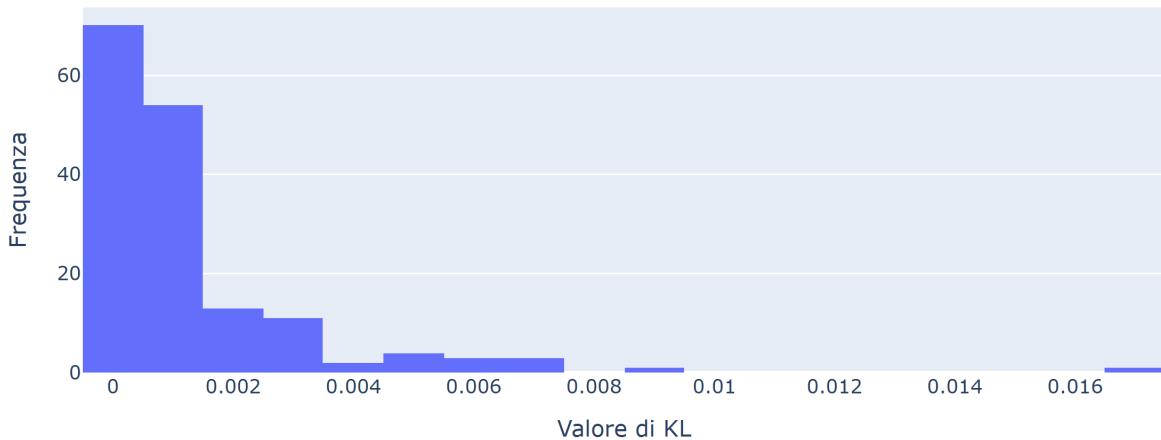


Figura 5.28: Grafici distribuzione |configurazione C, $M = 1$, input = 81|

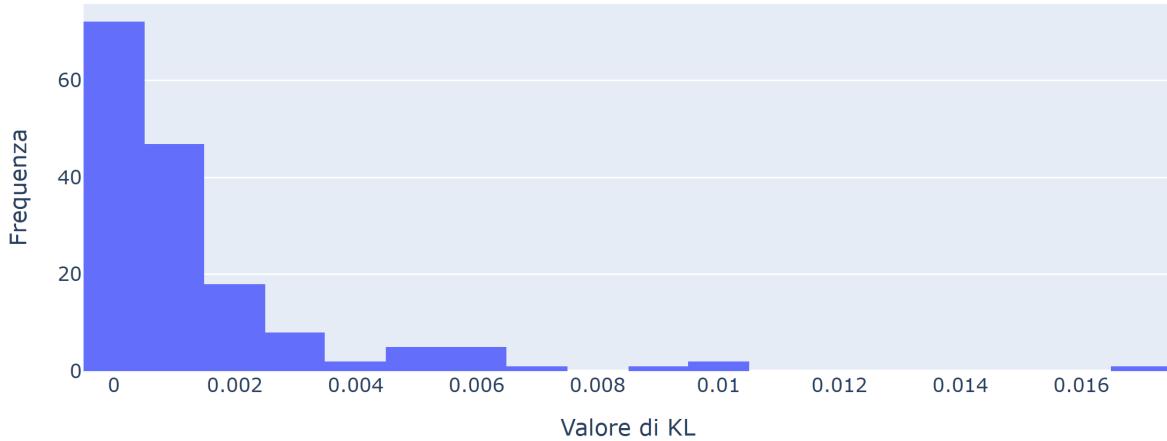


Figura 5.29: Grafici distribuzione |configurazione C, $M = 1$, input = 122|

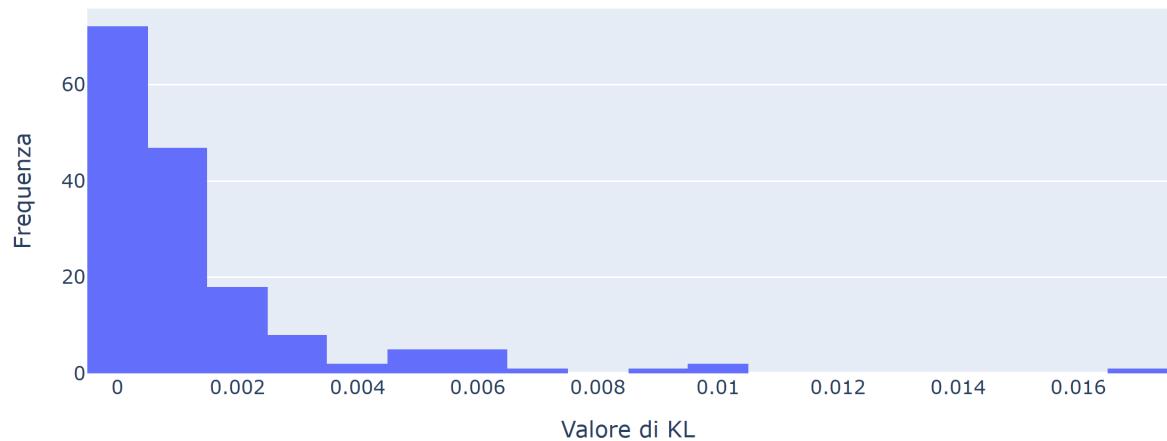


Figura 5.30: Grafici distribuzione |configurazione C, $M = 1$, input = 162|

Whisker Plot con la configurazione C e $M = 1$

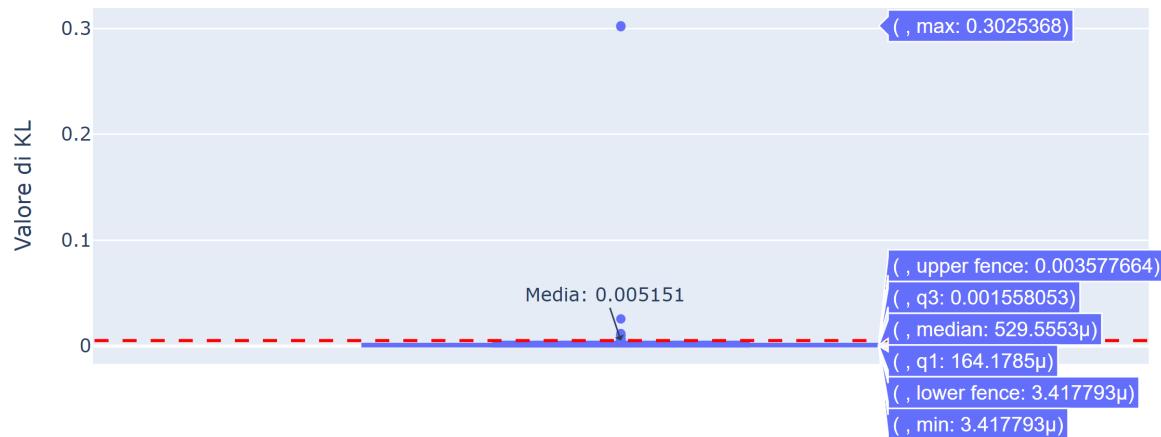


Figura 5.31: Whisker plot |configurazione C, $M = 1$, input = 8|



Figura 5.32: Whisker plot |configurazione C, $M = 1$, input = 16|

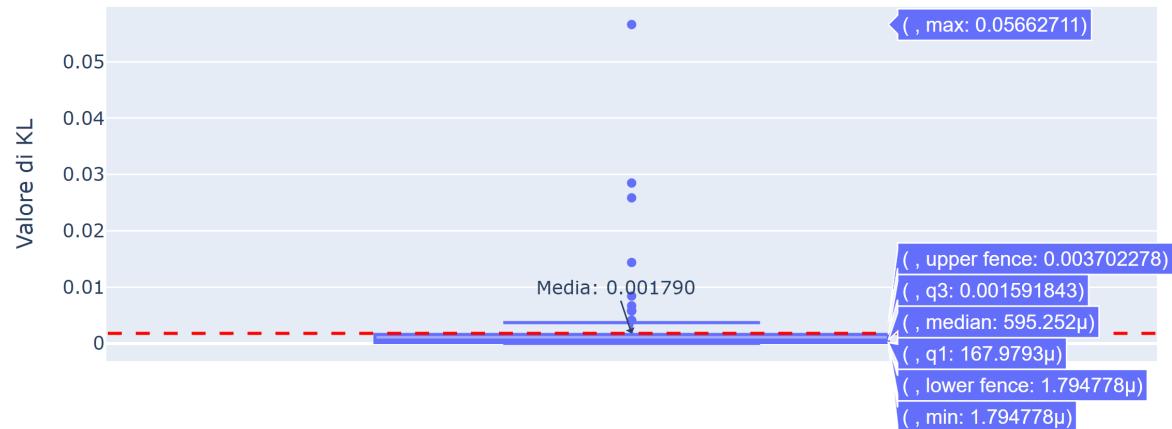


Figura 5.33: Whisker plot |configurazione C, $M = 1$, input = 40|

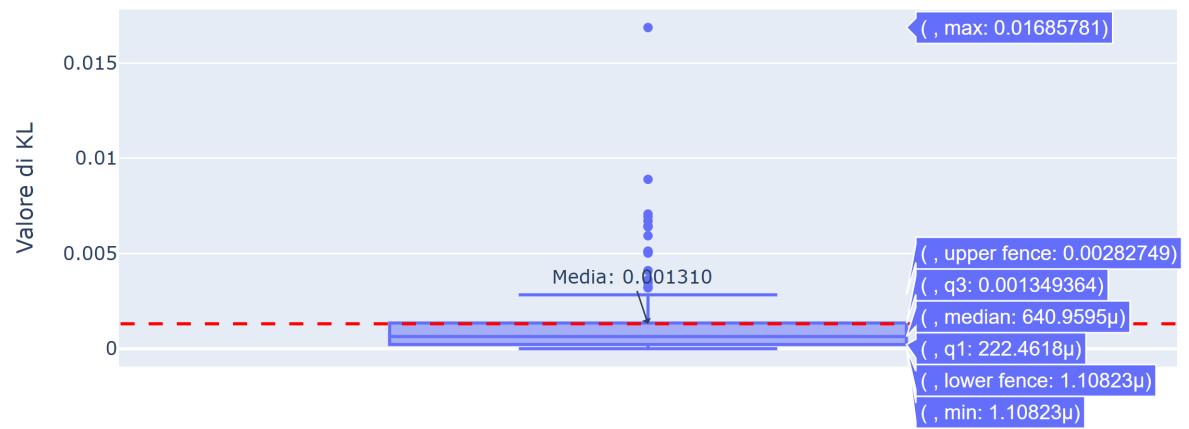


Figura 5.34: Whisker plot |configurazione C, $M = 1$, input = 81|

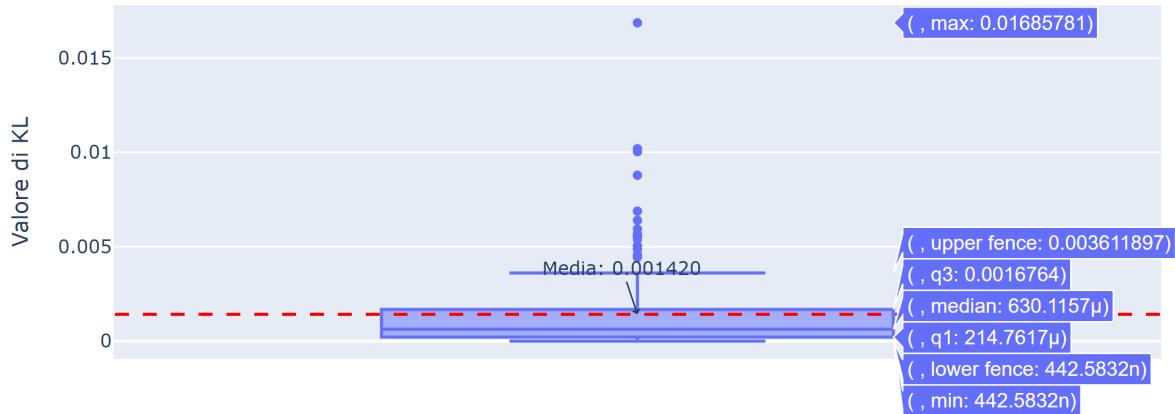


Figura 5.35: Whisker plot |configurazione C, $M = 1$, input = 122|

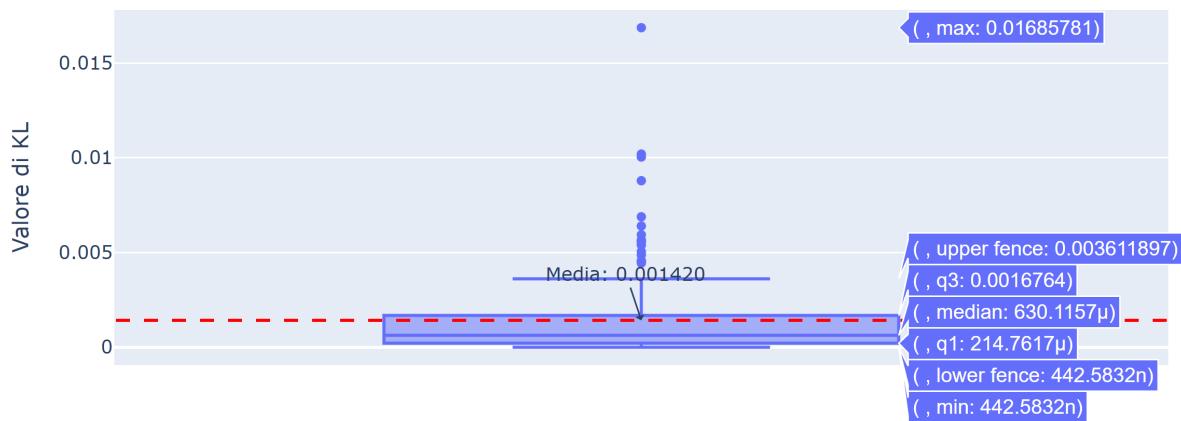


Figura 5.36: Whisker plot |configurazione C, $M = 1$, input = 162|

Distribuzione delle frequenze con la configurazione A e $M = 2$



Figura 5.37: Whisker plot |configurazione A, $M = 2$, input = 437|

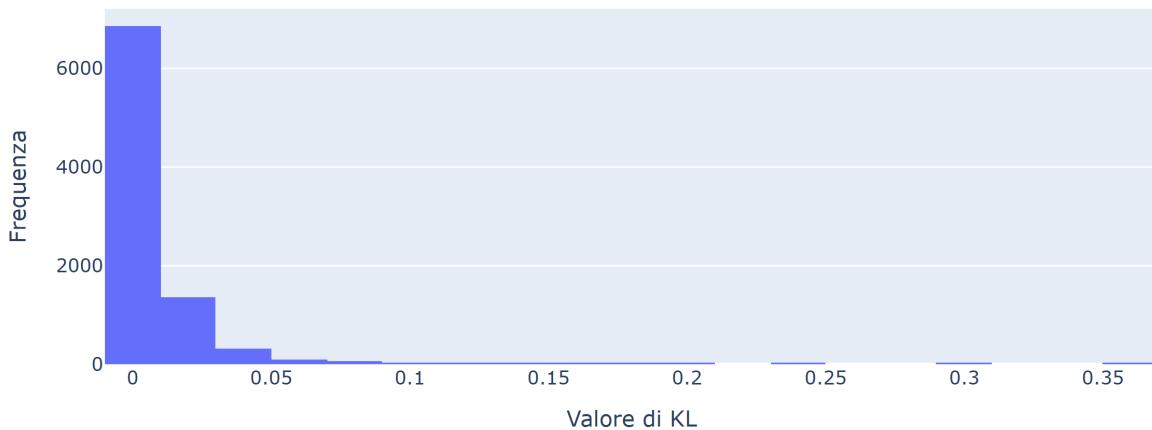


Figura 5.38: Whisker plot |configurazione A, $M = 2$, input = 875|

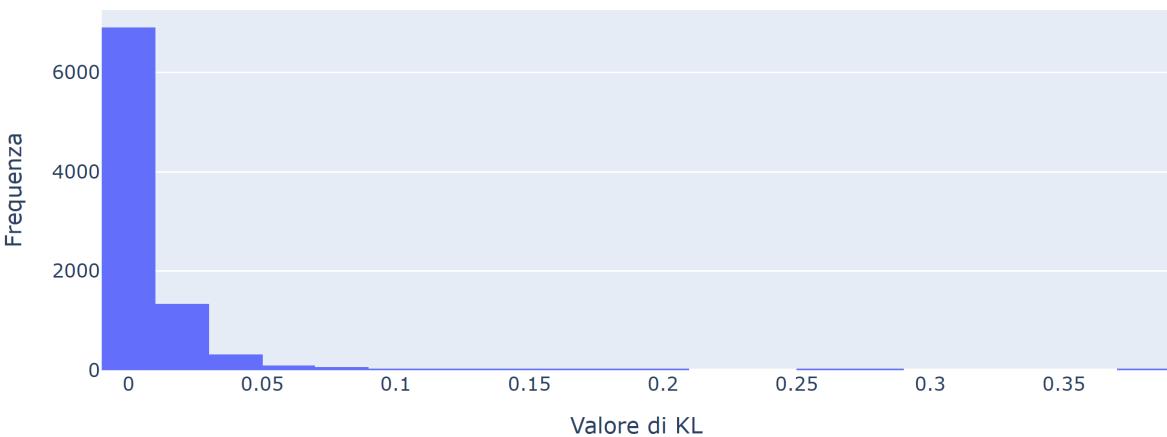


Figura 5.39: Whisker plot |configurazione A, $M = 2$, input = 2187|

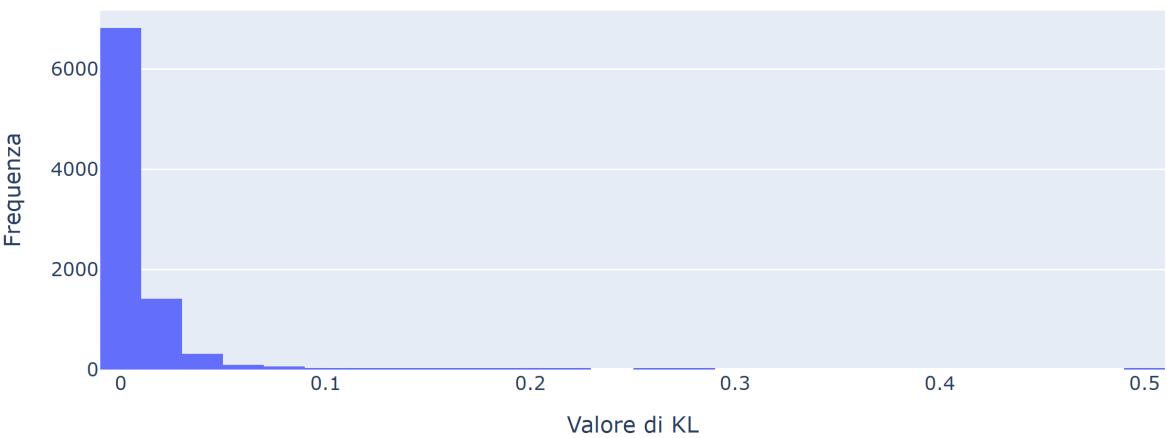


Figura 5.40: Whisker plot |configurazione A, $M = 2$, input = 4374|



Figura 5.41: Whisker plot |configurazione A, $M = 2$, input = 6561|

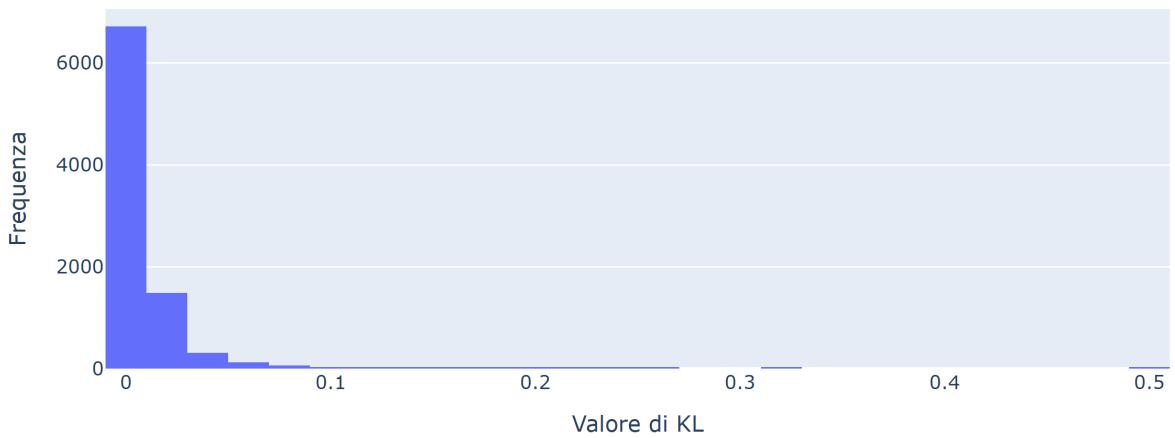


Figura 5.42: Whisker plot |configurazione A, $M = 2$, input = 8748|

Whisker Plot con la configurazione A e $M = 2$

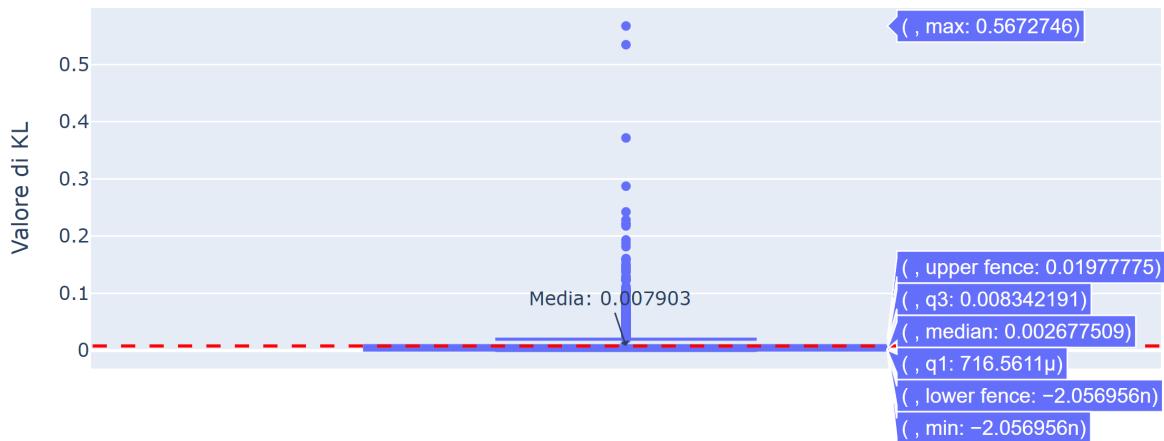


Figura 5.43: Whisker plot |configurazione A, $M = 2$, input = 437|

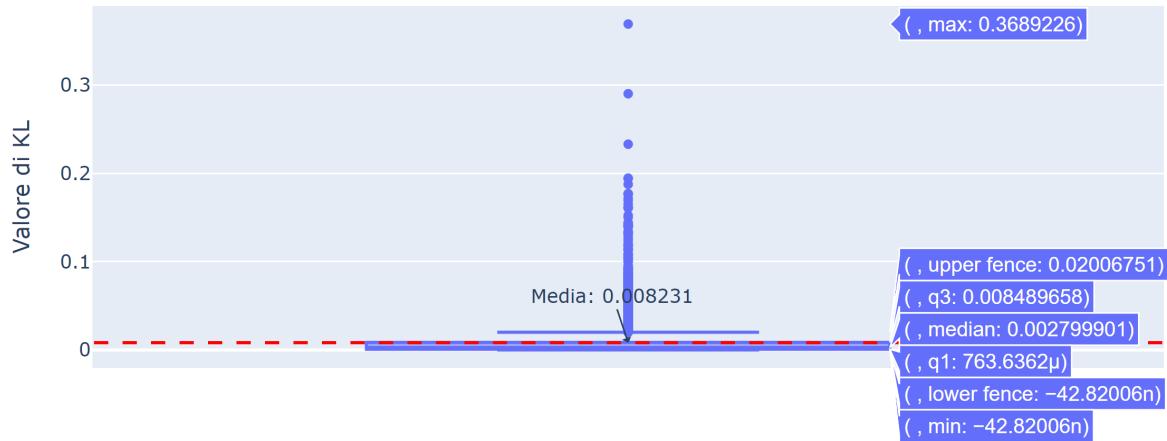


Figura 5.44: Whisker plot |configurazione A, $M = 2$, input = 875|

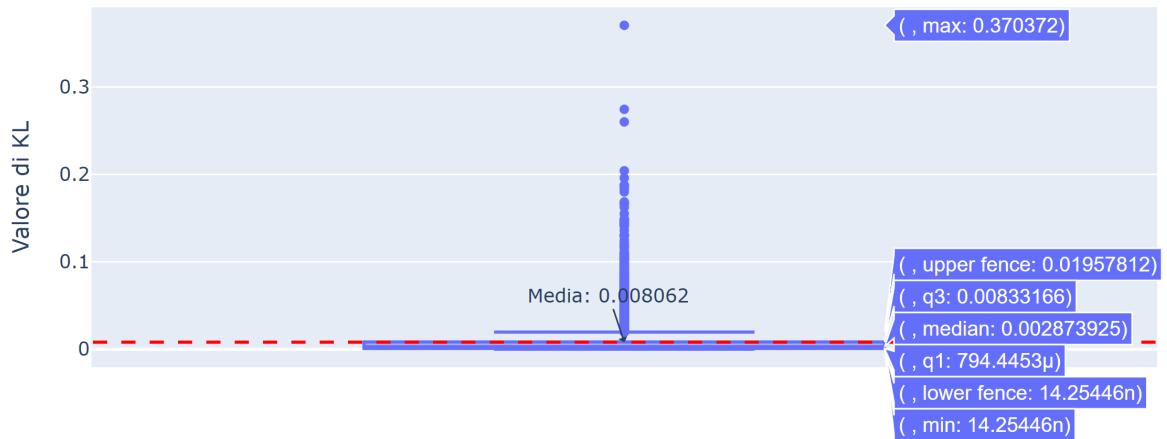


Figura 5.45: Whisker plot |configurazione A, $M = 2$, input = 2187|

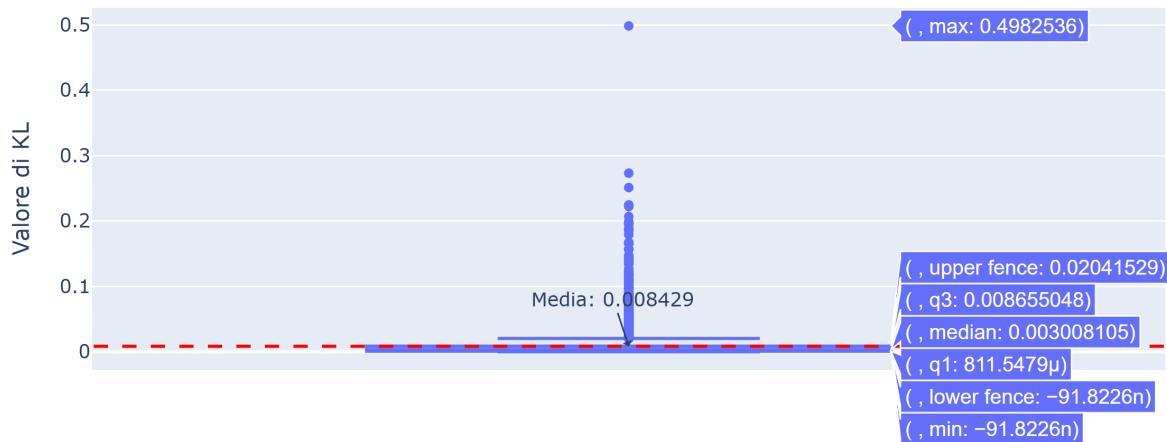


Figura 5.46: Whisker plot |configurazione A, $M = 2$, input = 4374|

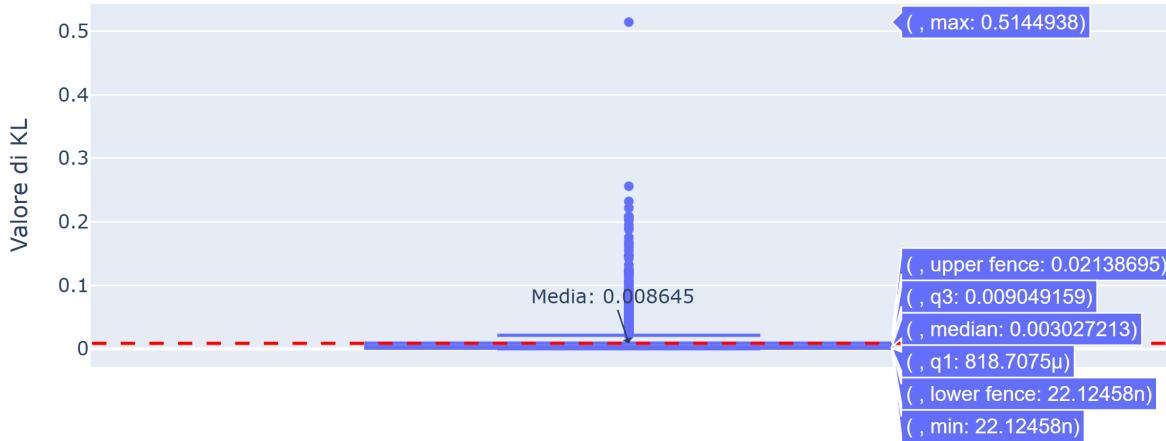


Figura 5.47: Whisker plot |configurazione A, $M = 2$, input = 6561|

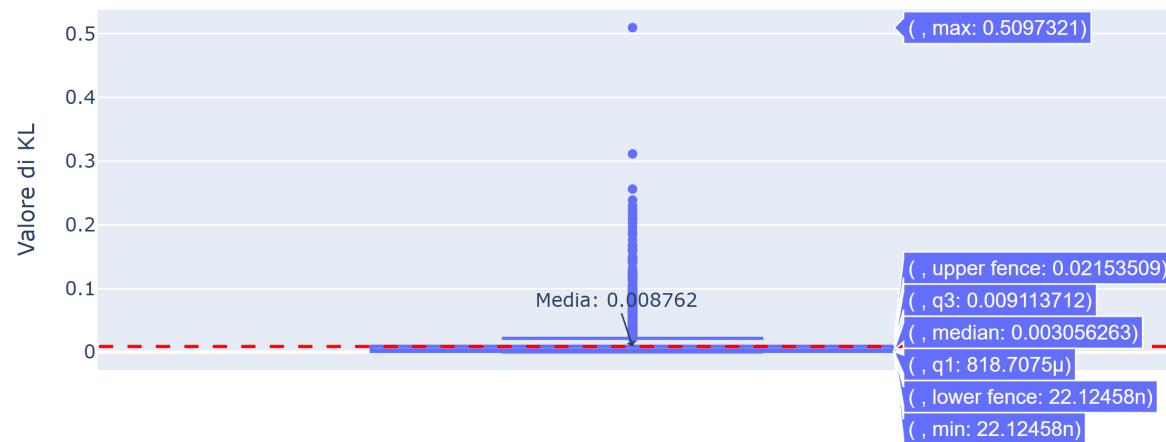


Figura 5.48: Whisker plot |configurazione A, $M = 2$, input = 8748|

Distribuzione delle frequenze con la configurazione B e $M = 2$

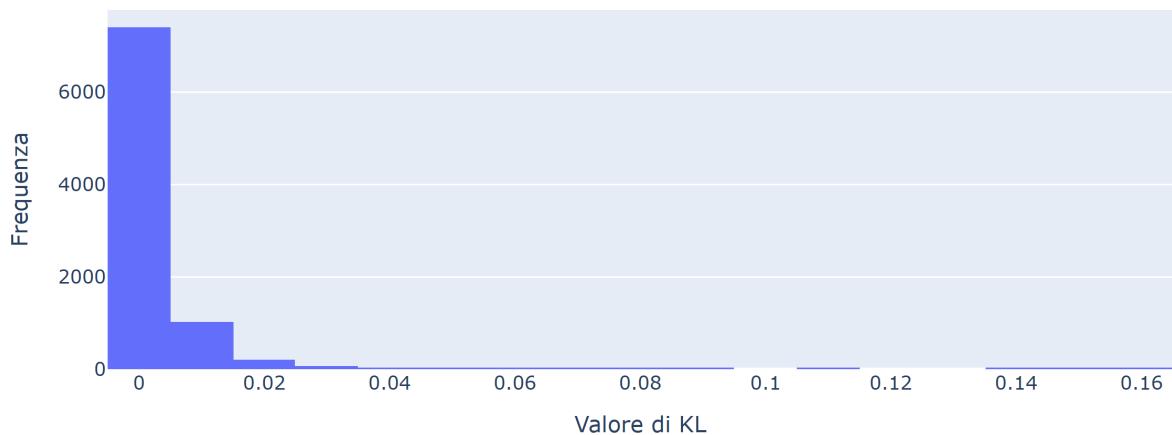


Figura 5.49: Grafici distribuzione |configurazione B, $M = 2$, input = 437|

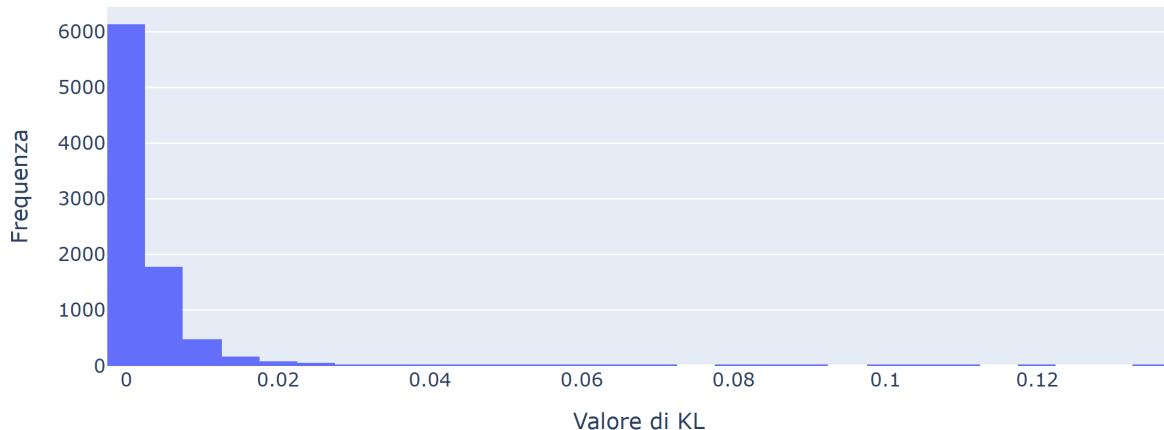


Figura 5.50: Grafici distribuzione |configurazione B, $M = 2$, input = 875|

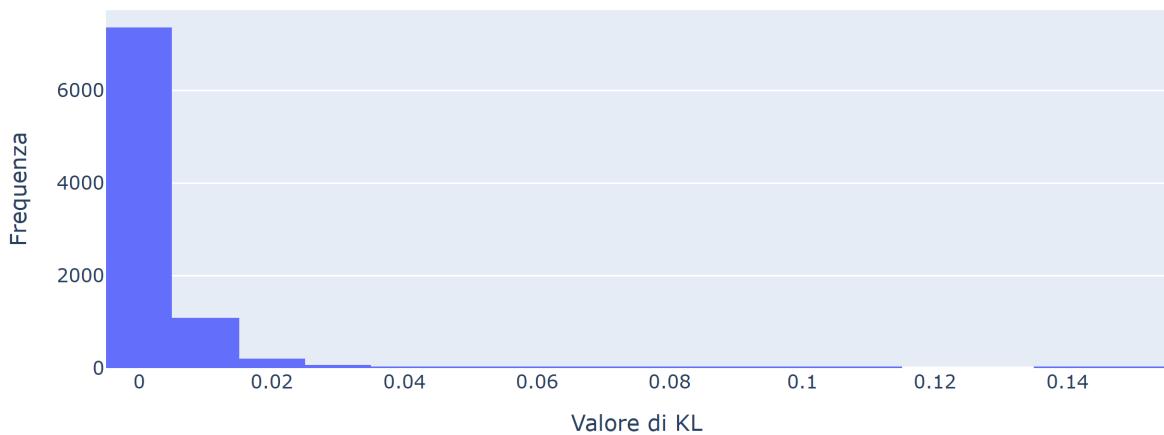


Figura 5.51: Grafici distribuzione |configurazione B, $M = 2$, input = 2187|



Figura 5.52: Grafici distribuzione |configurazione B, $M = 2$, input = 4374|

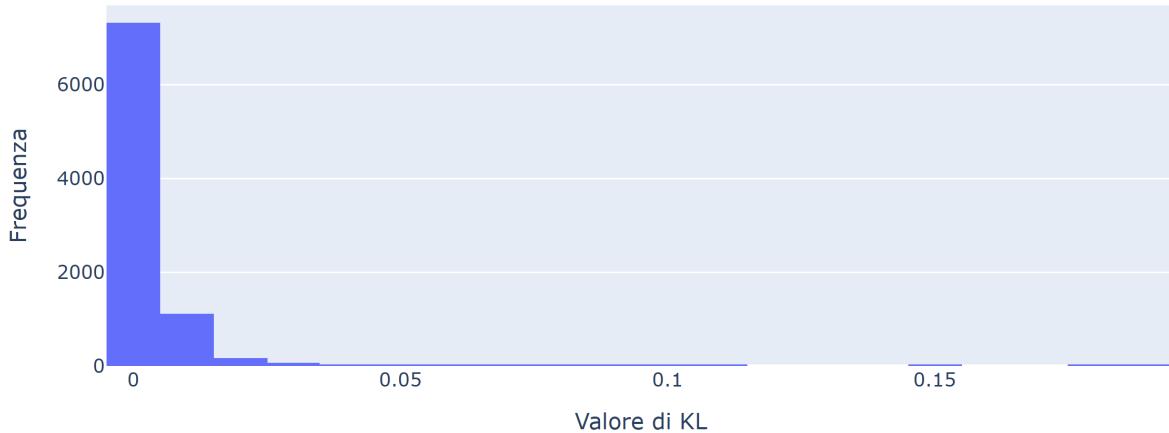


Figura 5.53: Grafici distribuzione |configurazione B, $M = 2$, input = 6561|



Figura 5.54: Grafici distribuzione |configurazione B, $M = 2$, input = 8748|

Whisker Plot con la configurazione B e $M = 2$

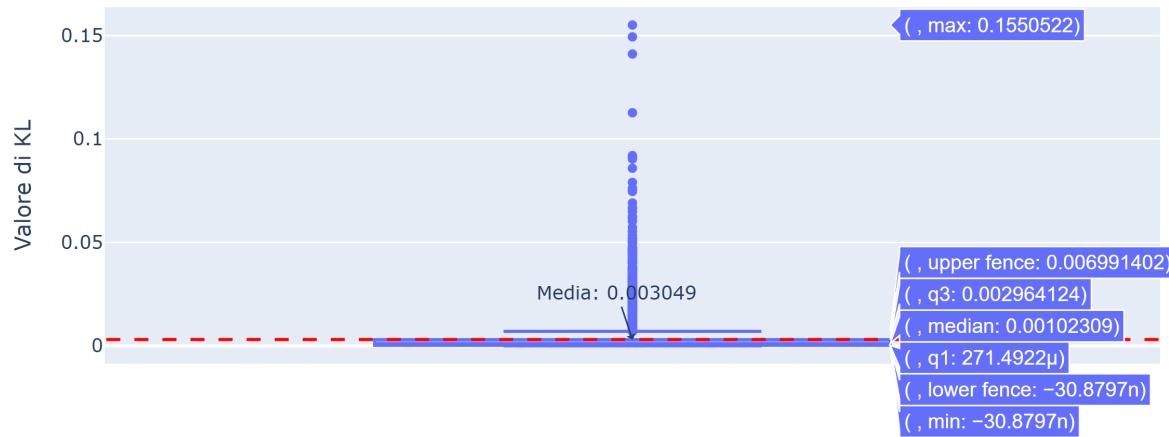


Figura 5.55: Whisker plot |configurazione B, $M = 2$, input = 437|

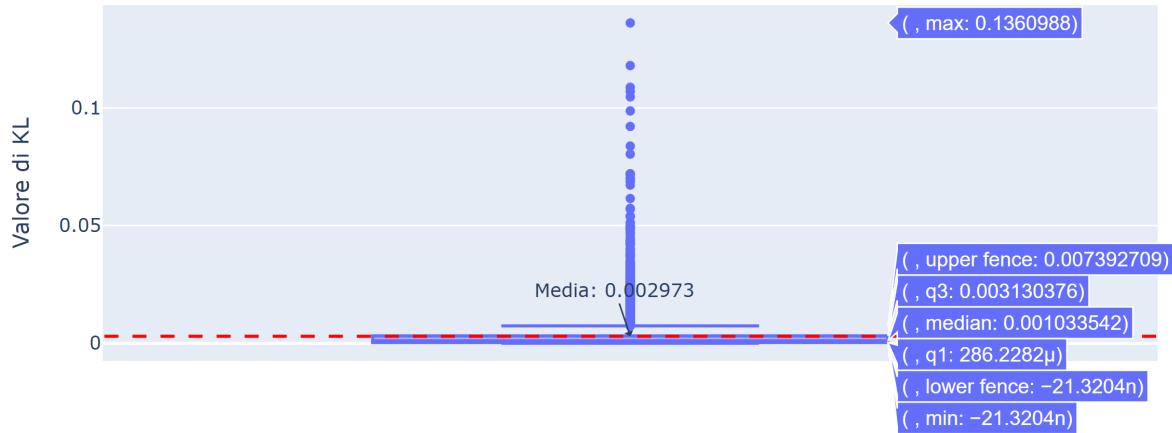


Figura 5.56: Whisker plot |configurazione B, $M = 2$, input = 875|

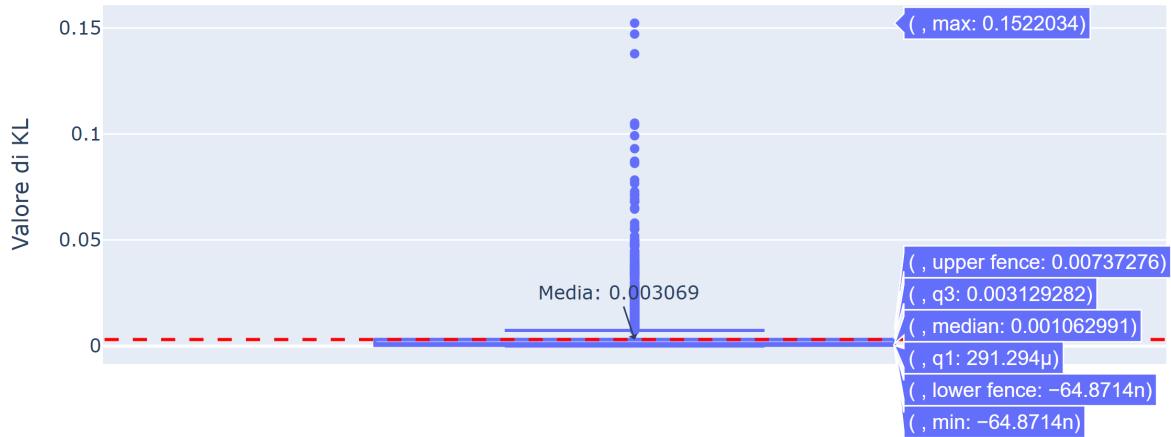


Figura 5.57: Whisker plot |configurazione B, $M = 2$, input = 2187|

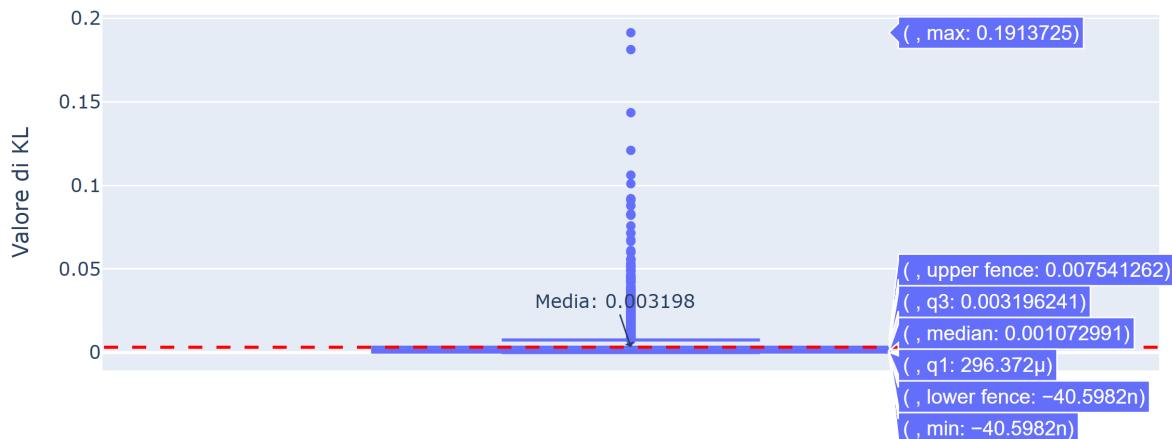


Figura 5.58: Whisker plot |configurazione B, $M = 2$, input = 4374|

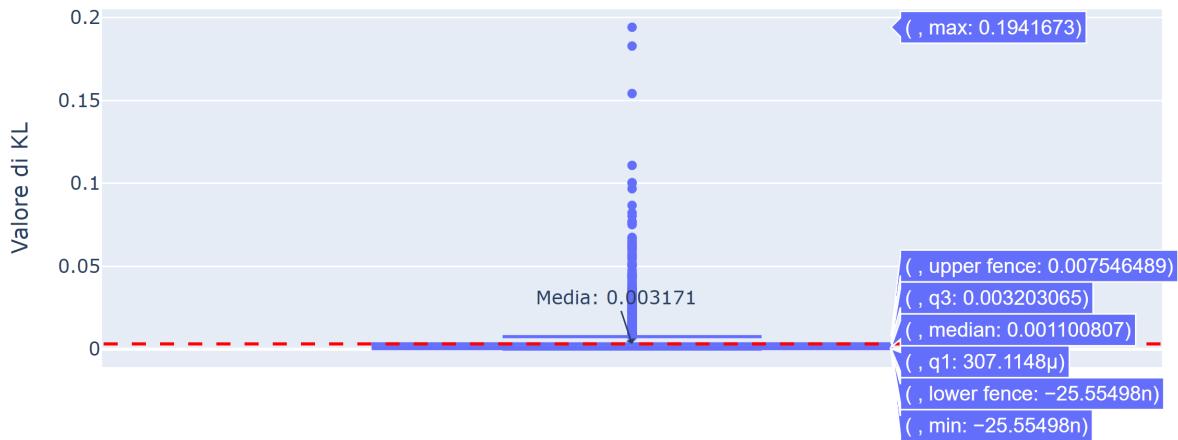


Figura 5.59: Whisker plot |configurazione B, $M = 2$, input = 6561|

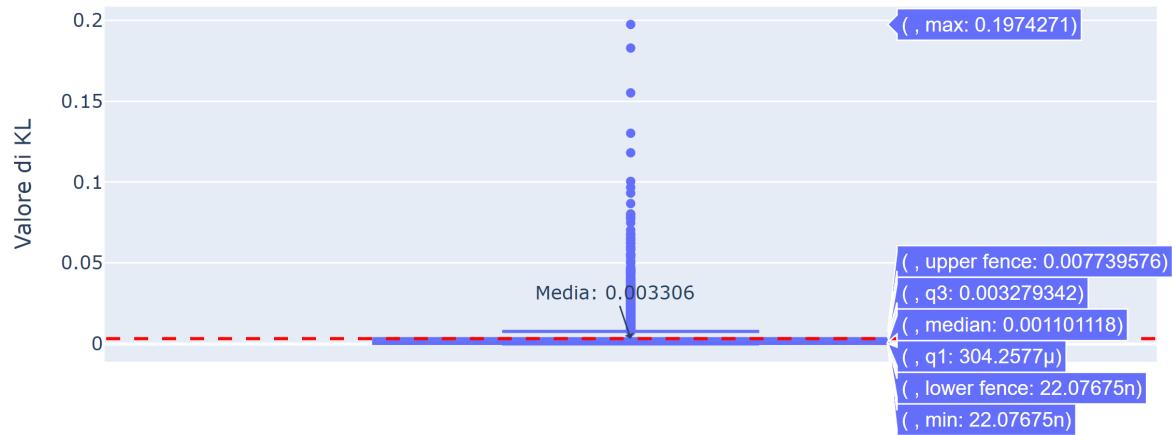


Figura 5.60: Whisker plot |configurazione B, $M = 2$, input = 8748|

Distribuzione delle frequenze con la configurazione C e $M = 2$



Figura 5.61: Grafici distribuzione |configurazione C, $M = 2$, input = 437|

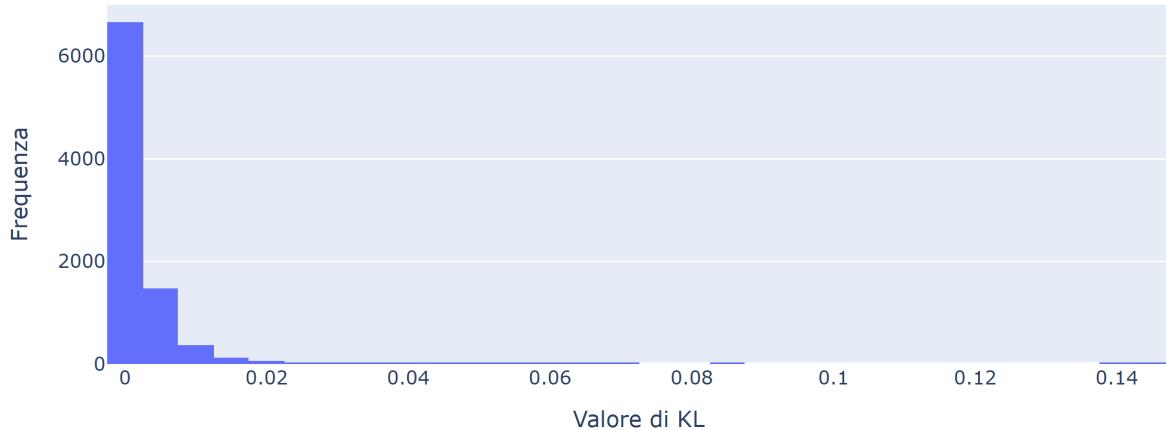


Figura 5.62: Grafici distribuzione |configurazione C, $M = 2$, input = 875|

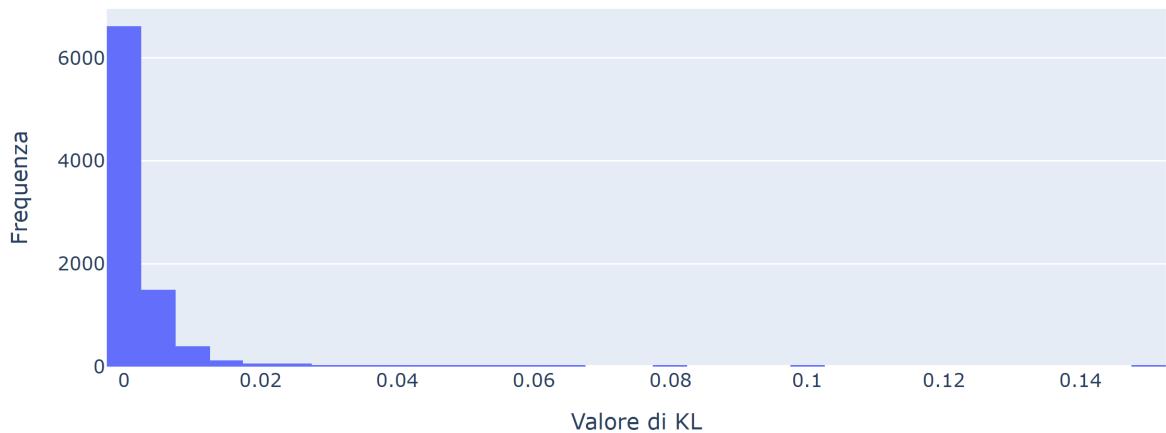


Figura 5.63: Grafici distribuzione |configurazione C, $M = 2$, input = 2187|

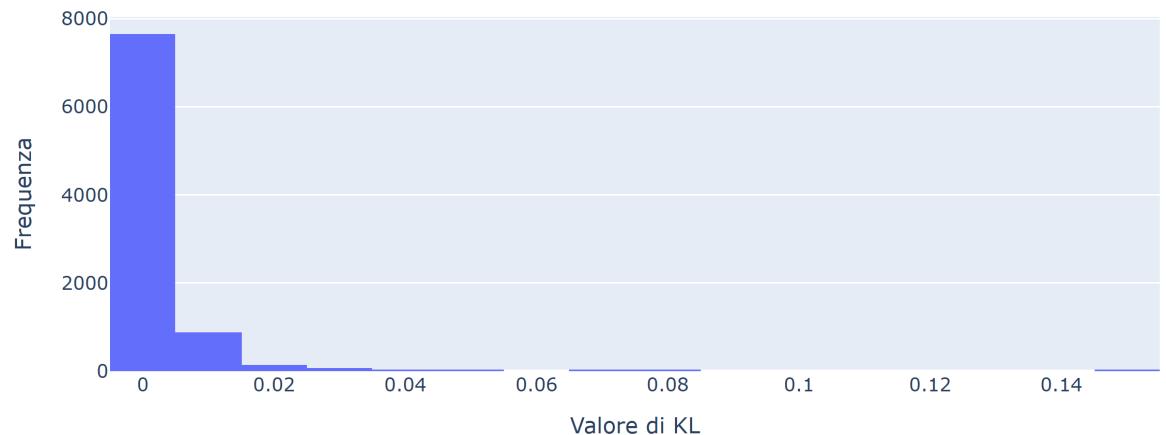


Figura 5.64: Grafici distribuzione |configurazione C, $M = 2$, input = 4374|

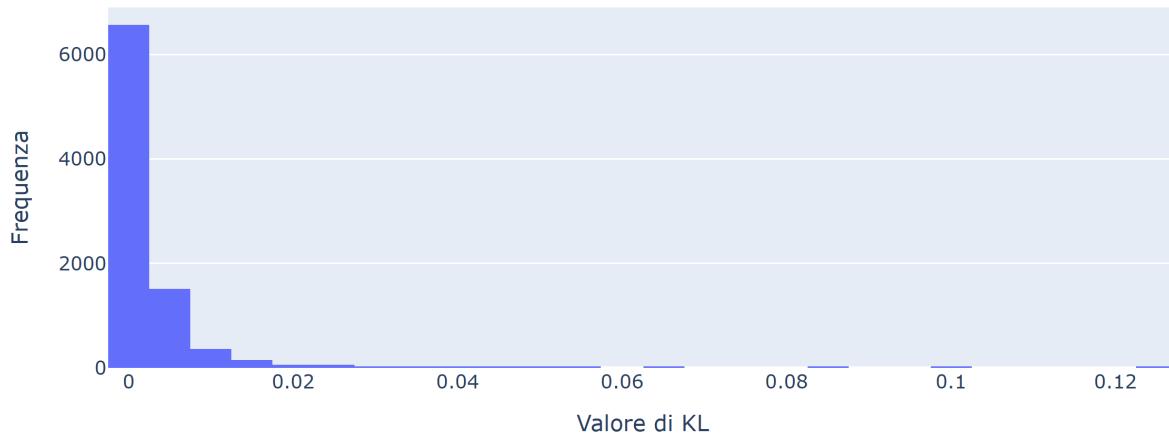


Figura 5.65: Grafici distribuzione |configurazione C, $M = 2$, input = 6561|

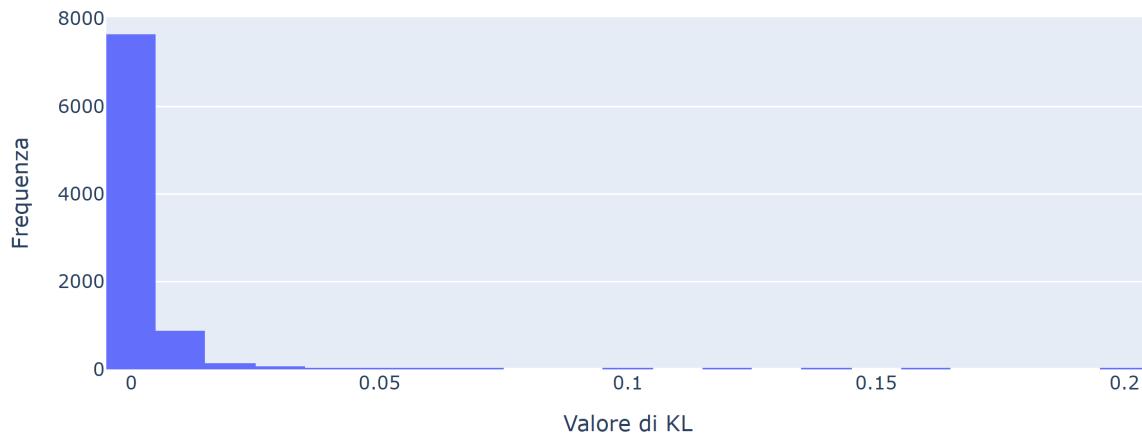


Figura 5.66: Grafici distribuzione |configurazione C, $M = 2$, input = 8748|

Whisker Plot con la configurazione C e $M = 2$

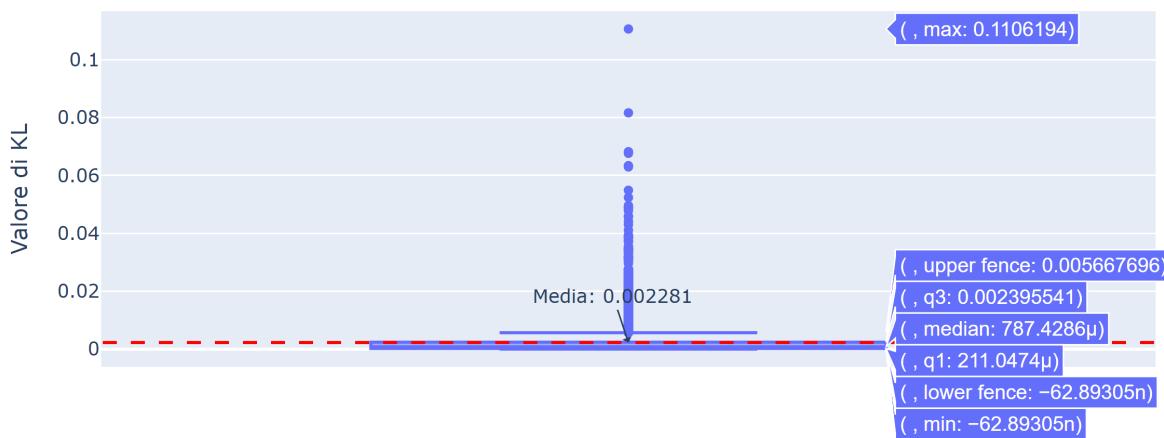


Figura 5.67: Whisker plot |configurazione C, $M = 2$, input = 437|

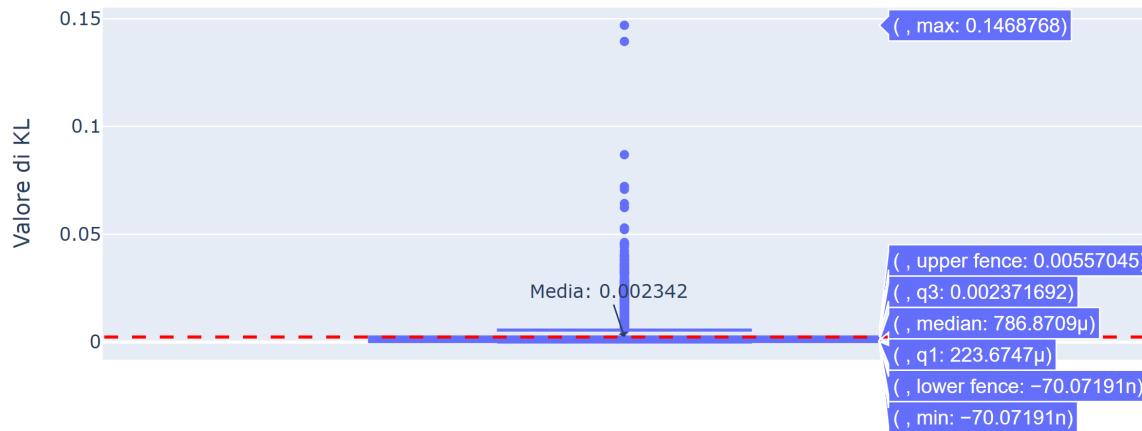


Figura 5.68: Whisker plot |configurazione C, $M = 2$, input = 875|

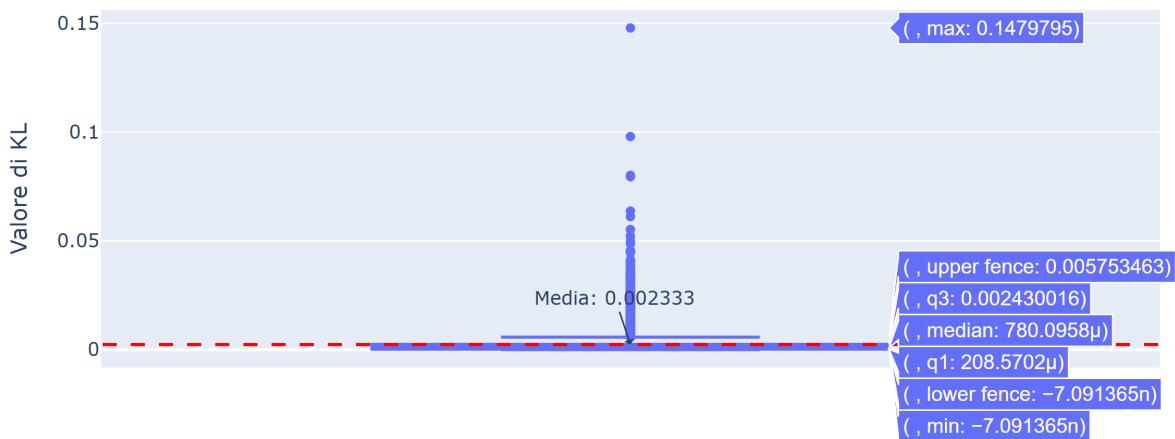


Figura 5.69: Whisker plot |configurazione C, $M = 2$, input = 2187|

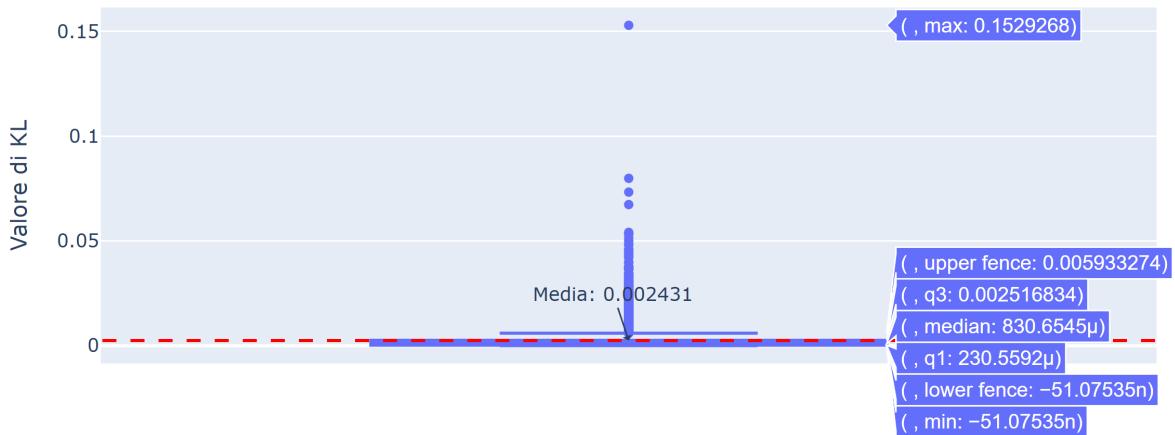


Figura 5.70: Whisker plot |configurazione C, $M = 2$, input = 4374|

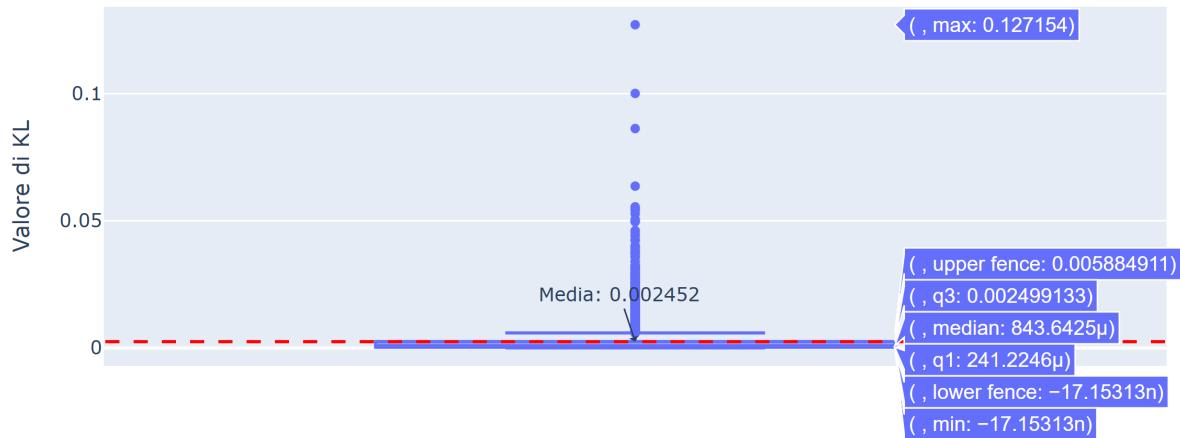


Figura 5.71: Whisker plot |configurazione C, $M = 2$, input = 6561|

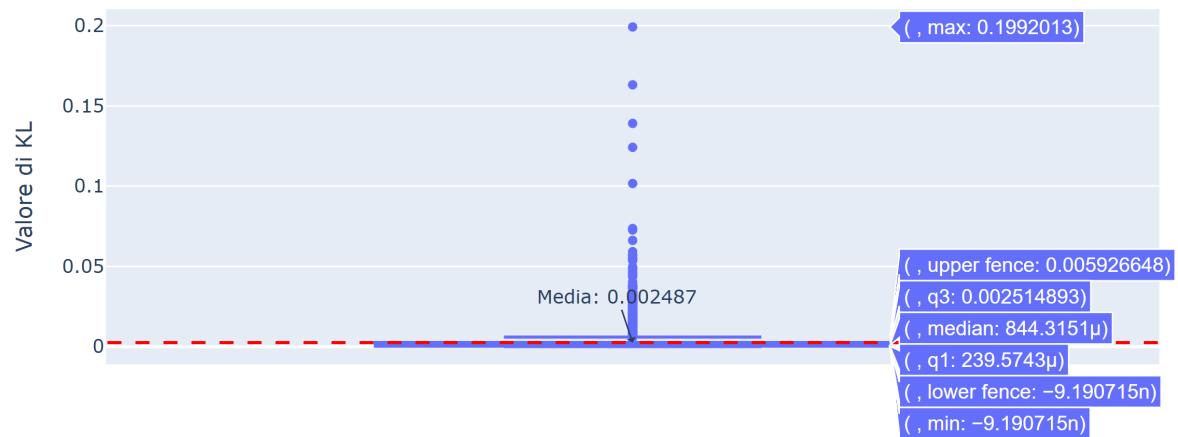


Figura 5.72: Grafici distribuzione |configurazione C, $M = 2$, input = 23620|

Distribuzione delle frequenze con la configurazione A e $M = 3$

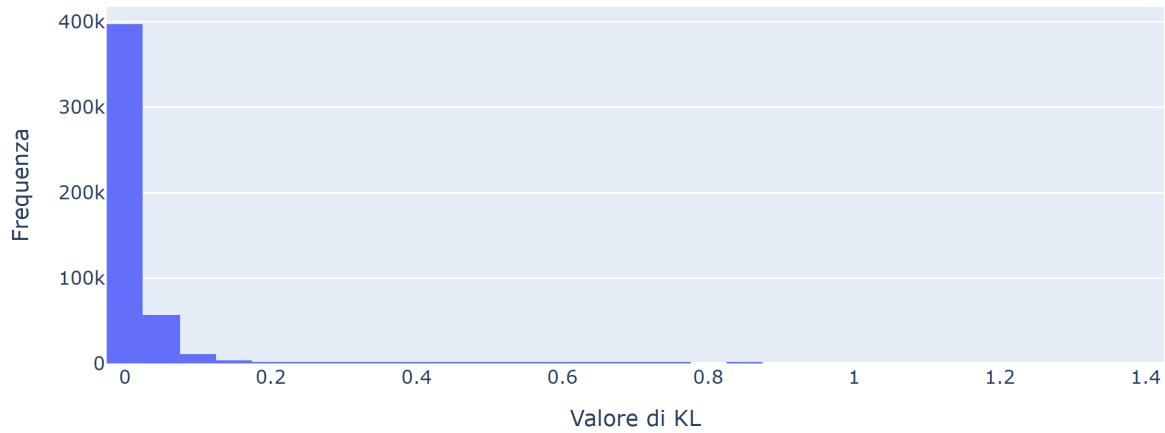


Figura 5.73: Grafici distribuzione |configurazione A, $M = 3$, input = 23620|

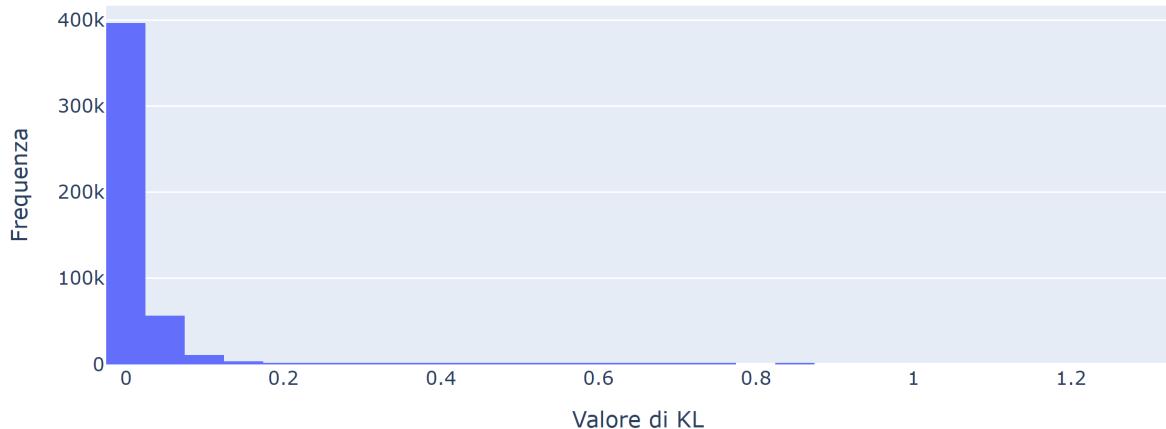


Figura 5.74: Grafici distribuzione |configurazione A, $M = 3$, input = 47239|

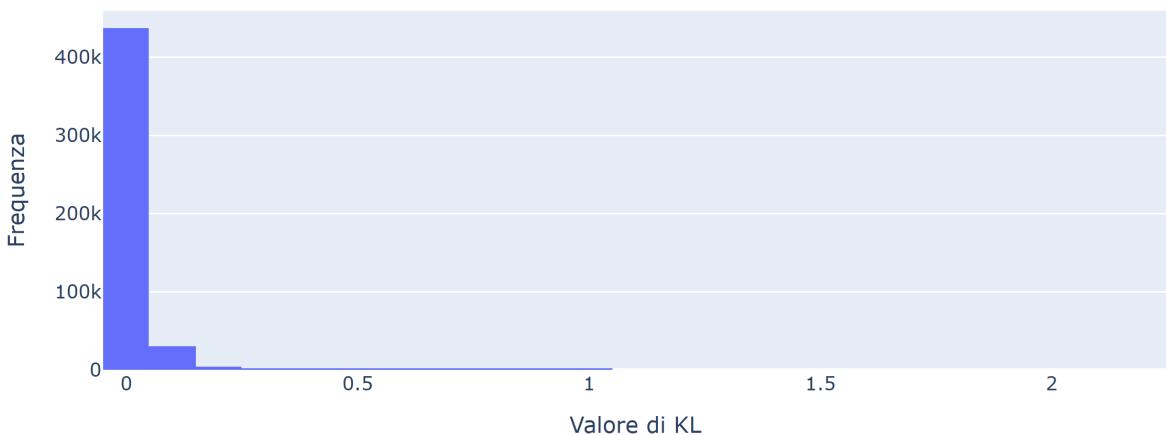


Figura 5.75: Grafici distribuzione |configurazione A, $M = 3$, input = 118098|

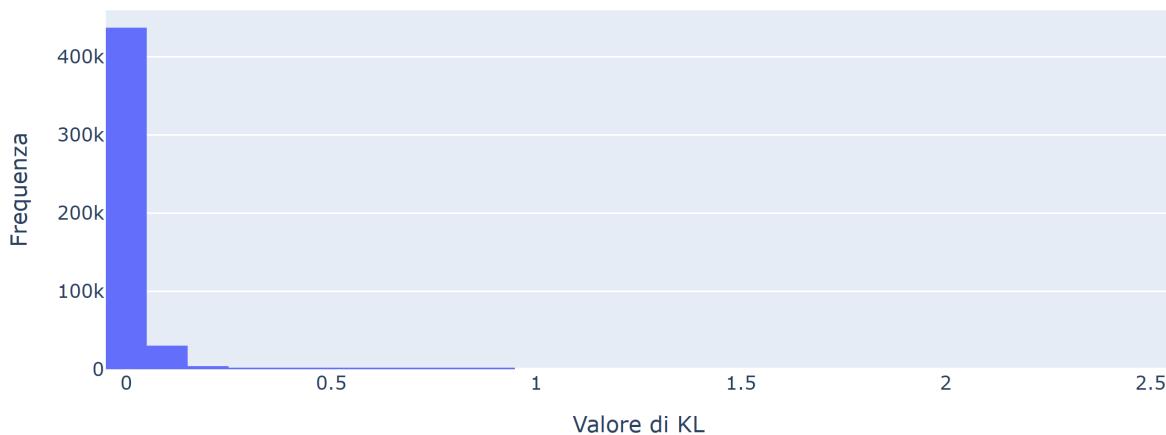


Figura 5.76: Grafici distribuzione |configurazione A, $M = 3$, input = 236196|

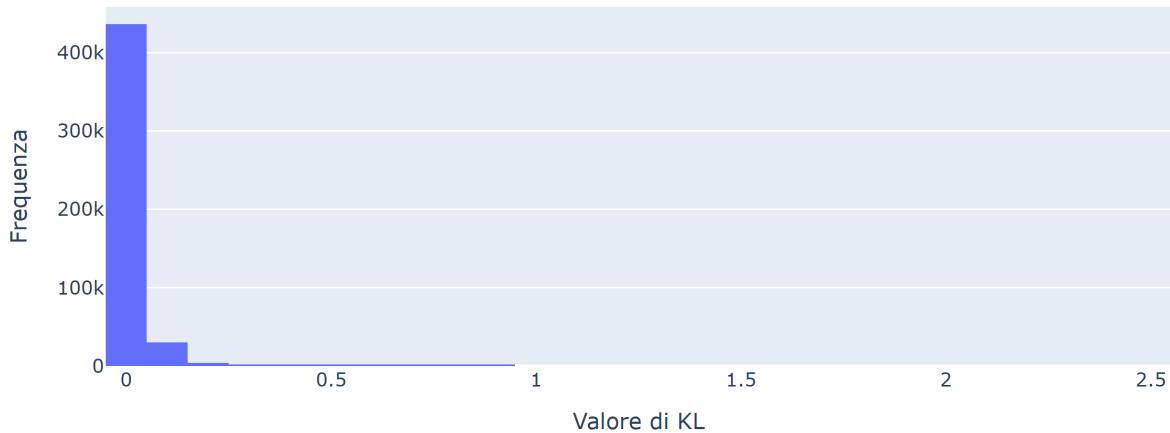


Figura 5.77: Grafici distribuzione |configurazione A, $M = 3$, input = 354294|



Figura 5.78: Grafici distribuzione |configurazione A, $M = 3$, input = 472392|

Whisker Plot con la configurazione A e $M = 3$

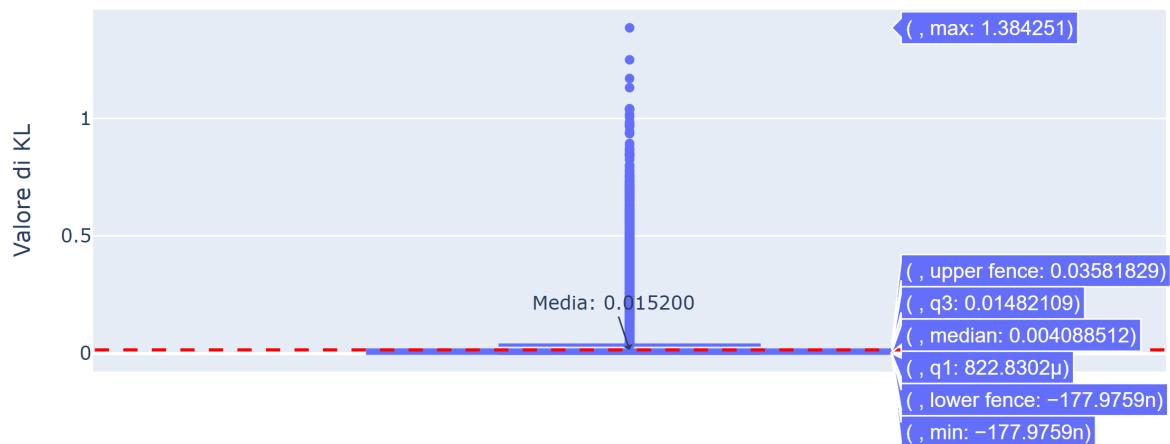


Figura 5.79: Whisker plot |configurazione A, $M = 3$, input = 23620|

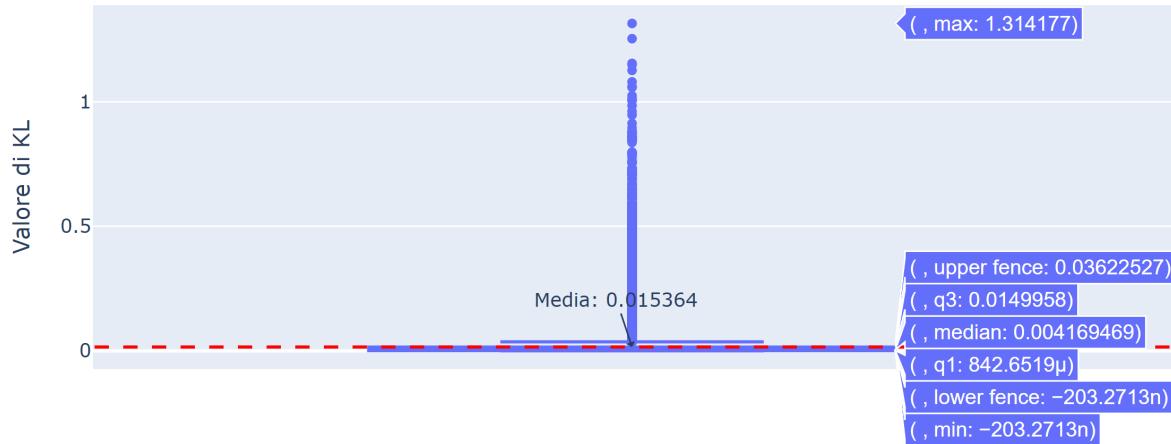


Figura 5.80: Whisker plot |configurazione A, $M = 3$, input = 47239|

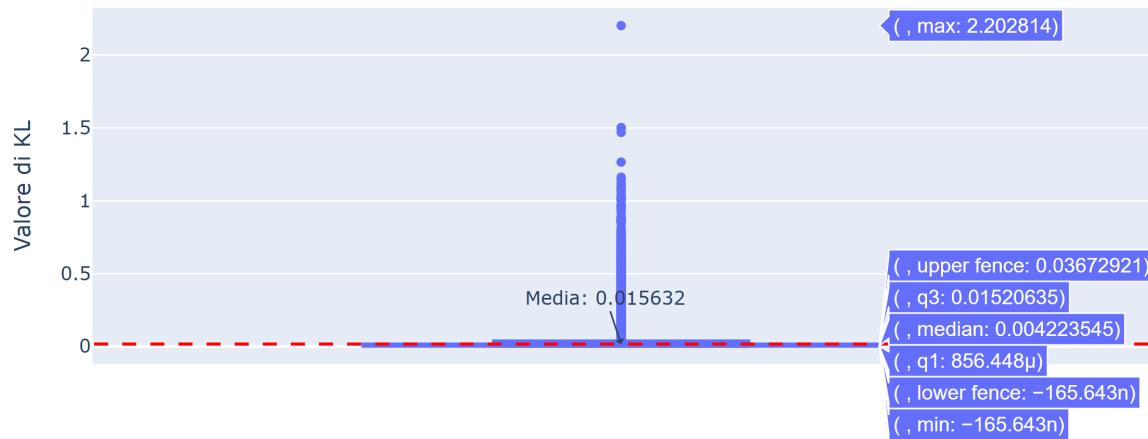


Figura 5.81: Whisker plot |configurazione A, $M = 3$, input = 118098|

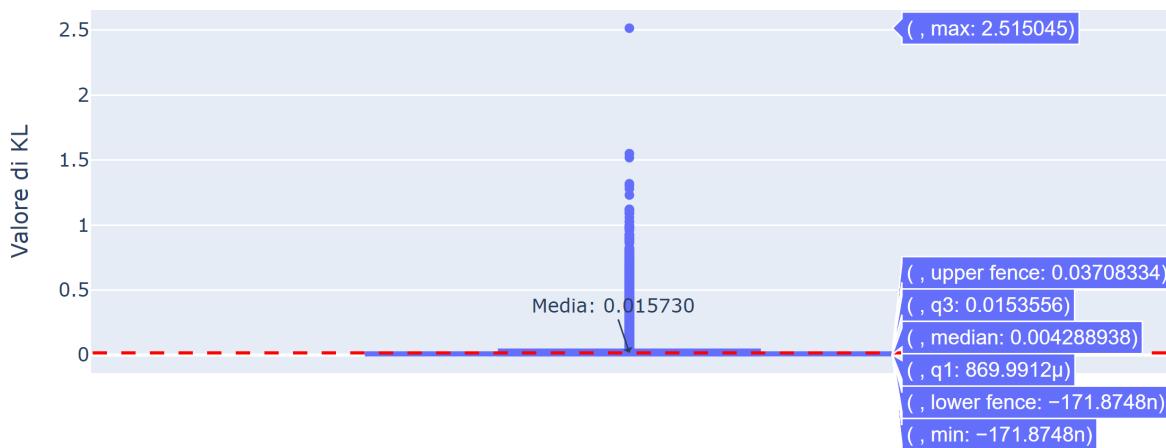


Figura 5.82: Whisker plot |configurazione A, $M = 3$, input = 236196|

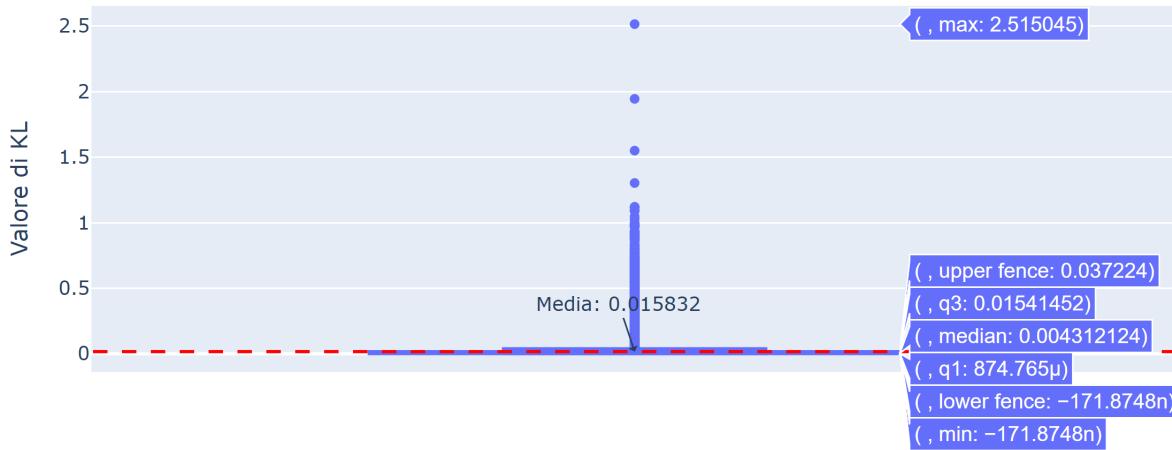


Figura 5.83: Whisker plot |configurazione A, $M = 3$, input = 354294|

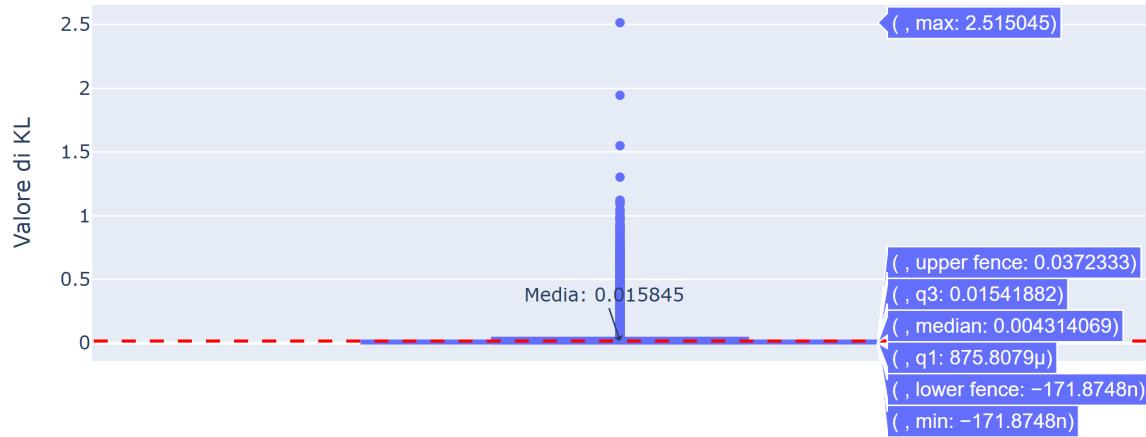


Figura 5.84: Whisker plot |configurazione A, $M = 3$, input = 472392|

Distribuzione delle frequenze con la configurazione B e $M = 3$

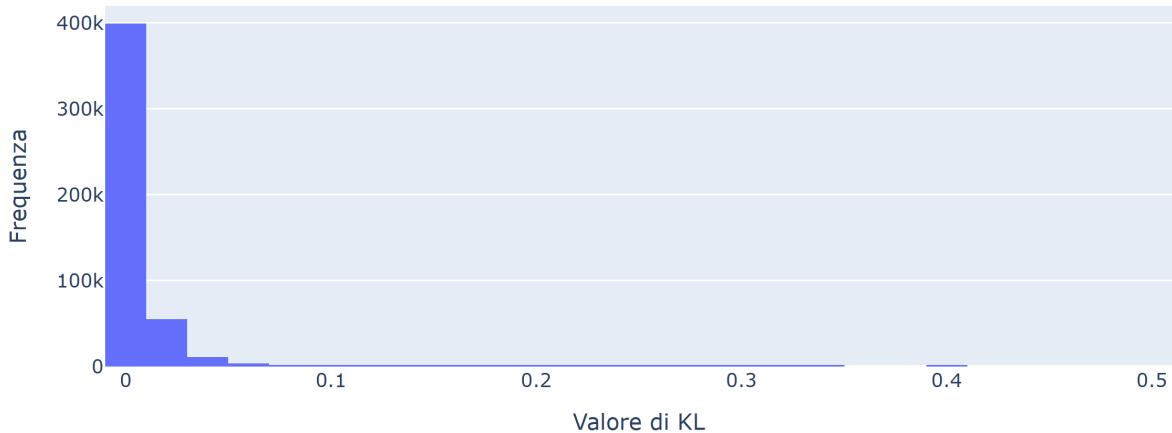


Figura 5.85: Grafici distribuzione |configurazione B, $M = 3$, input = 23620|

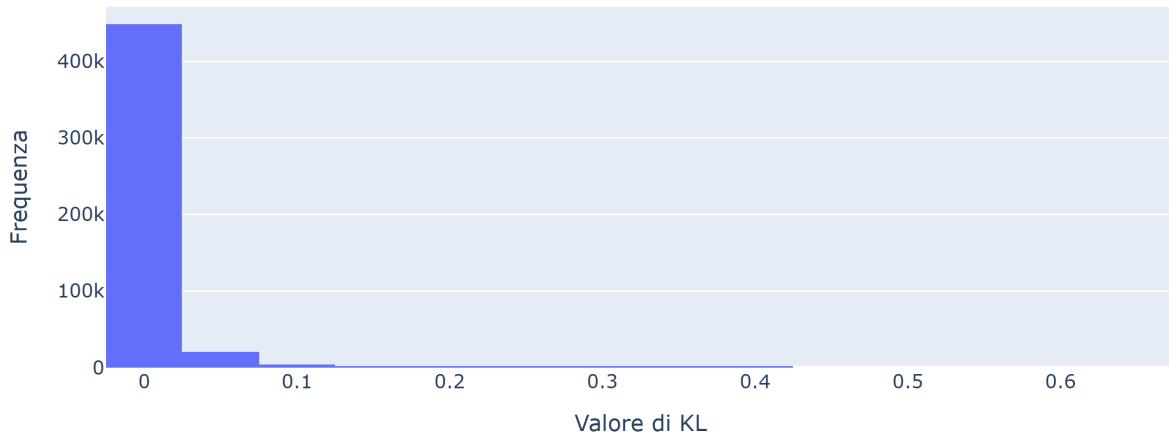


Figura 5.86: Grafici distribuzione |configurazione B, $M = 3$, input = 47239|

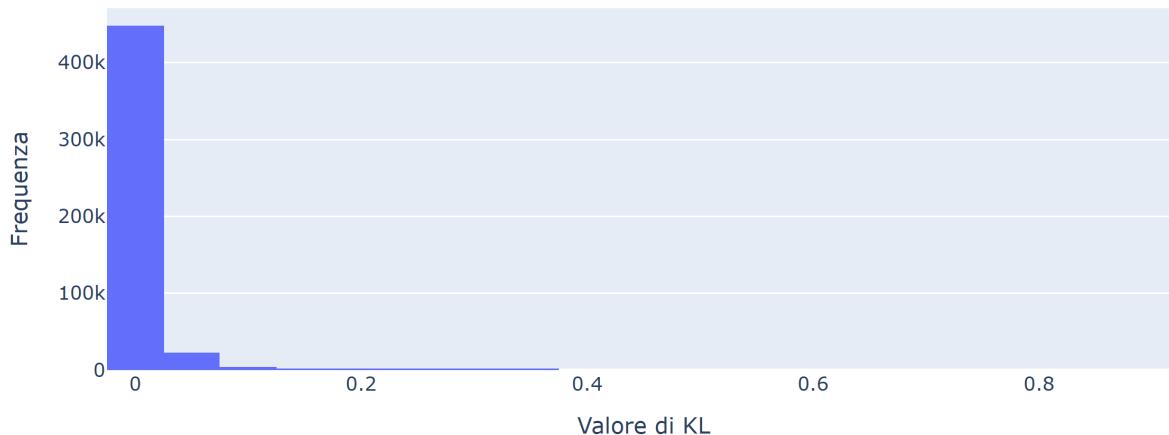


Figura 5.87: Grafici distribuzione |configurazione B, $M = 3$, input = 118098|

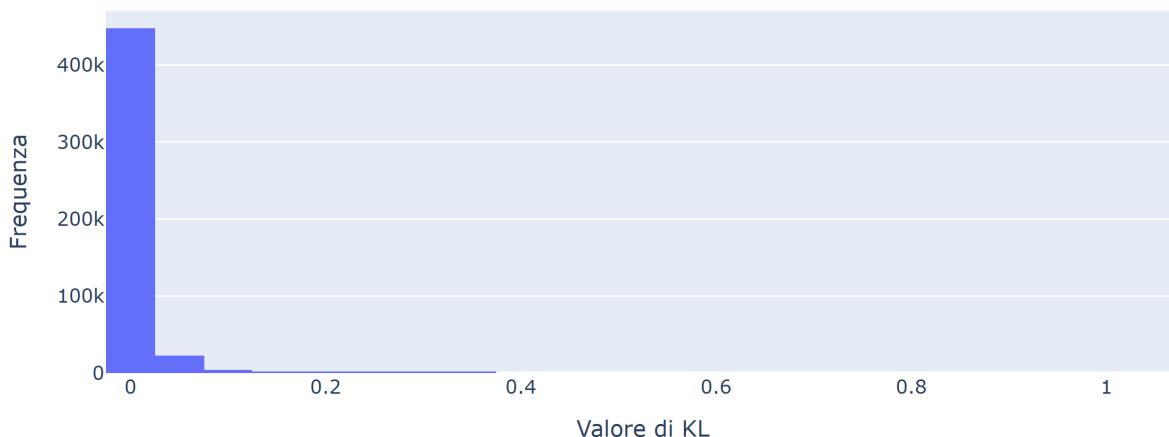


Figura 5.88: Grafici distribuzione |configurazione B, $M = 3$, input = 236196|

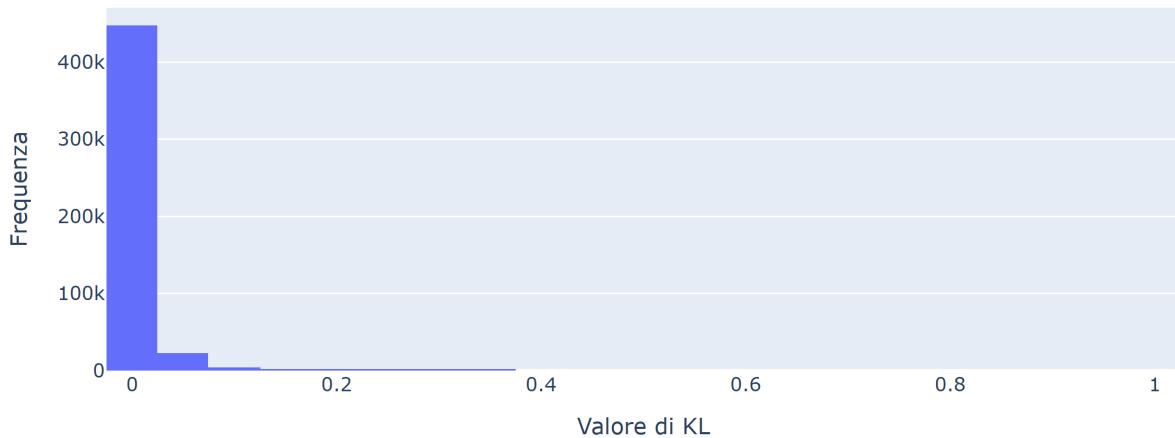


Figura 5.89: Grafici distribuzione |configurazione B, $M = 3$, input = 354294|

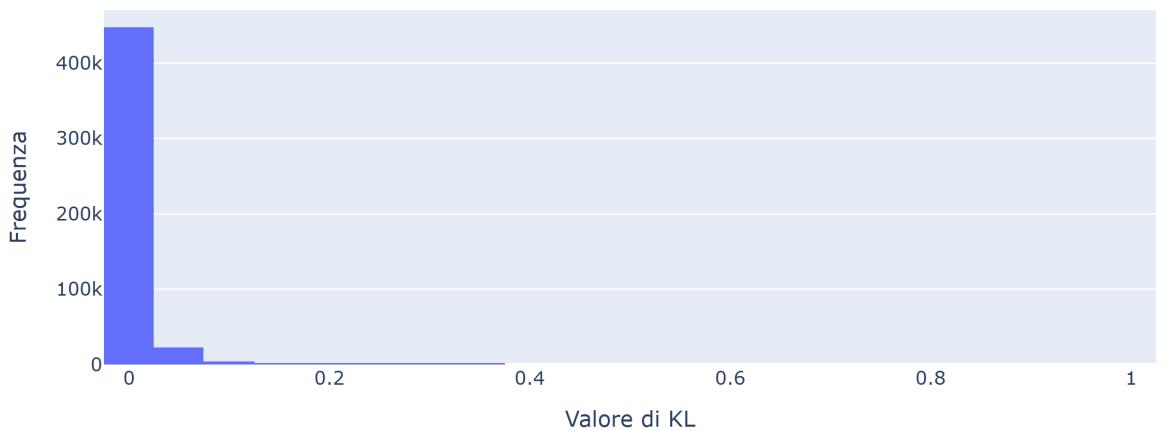


Figura 5.90: Grafici distribuzione |configurazione B, $M = 3$, input = 472392|

Whisker Plot con la configurazione B e $M = 3$

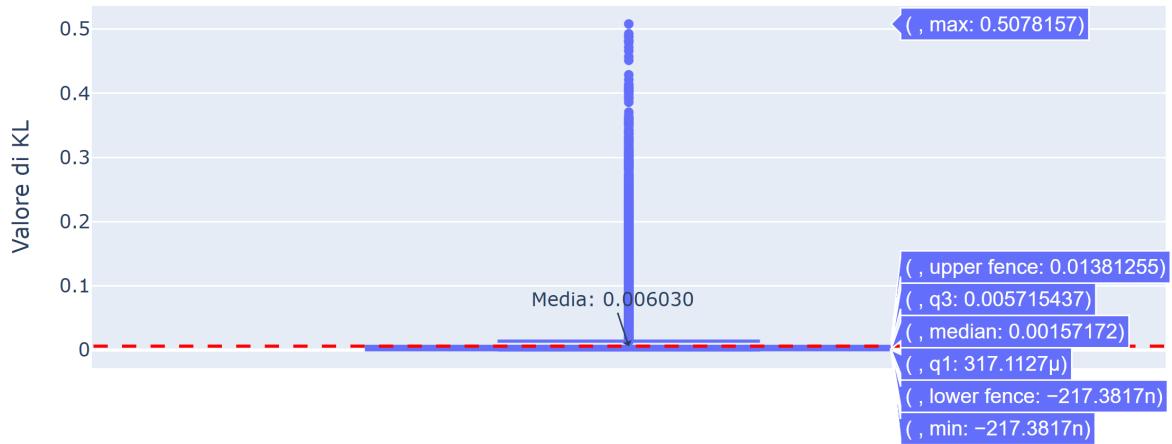


Figura 5.91: Whisker plot |configurazione B, $M = 3$, input = 23260|

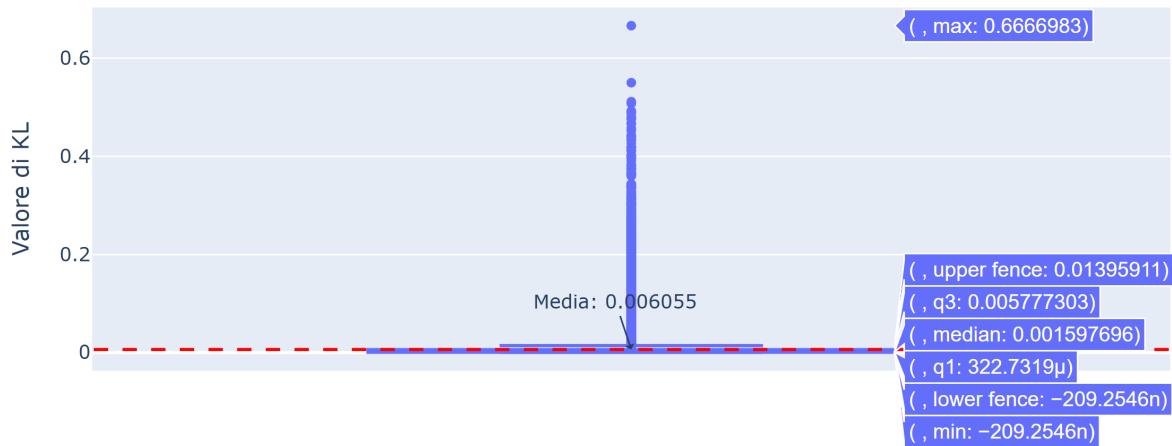


Figura 5.92: Whisker plot |configurazione B, $M = 3$, input = 47239|

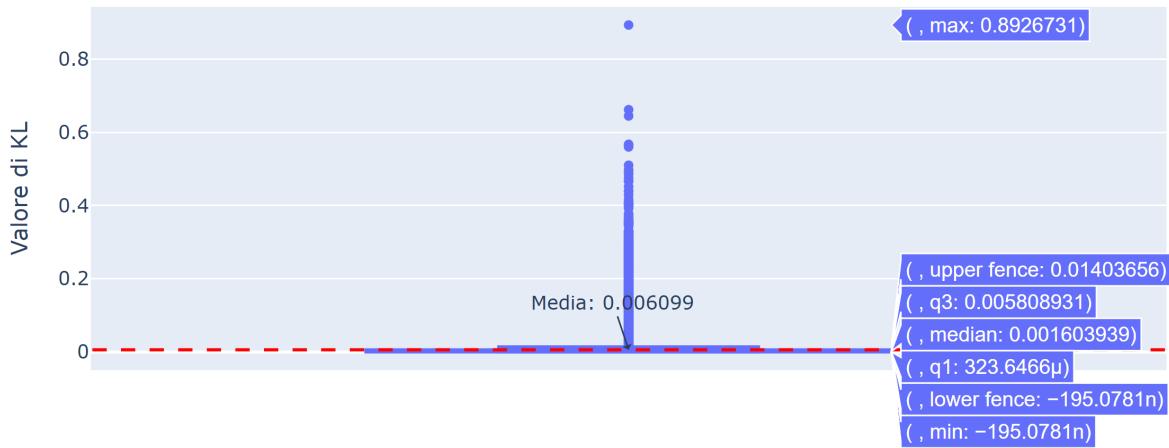


Figura 5.93: Whisker plot |configurazione B, $M = 3$, input = 118098|

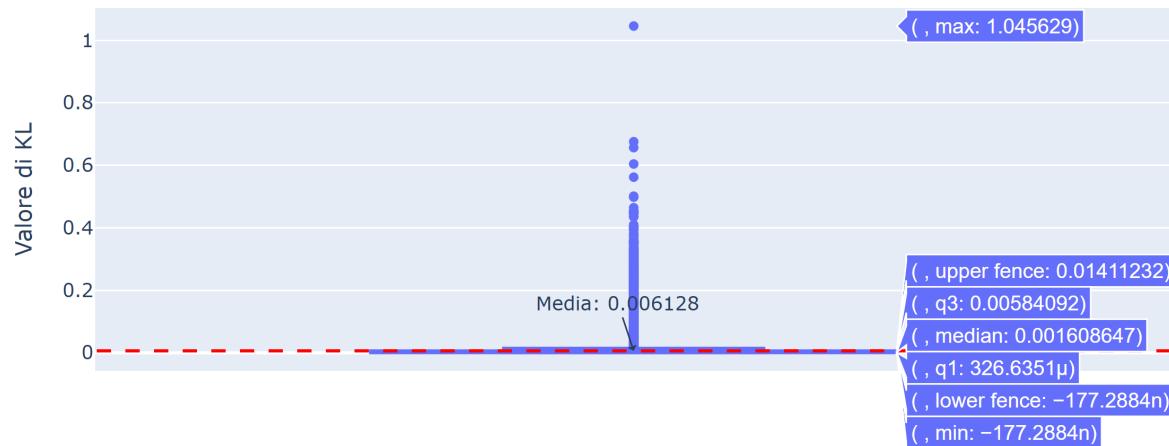


Figura 5.94: Whisker plot |configurazione B, $M = 3$, input = 236196|

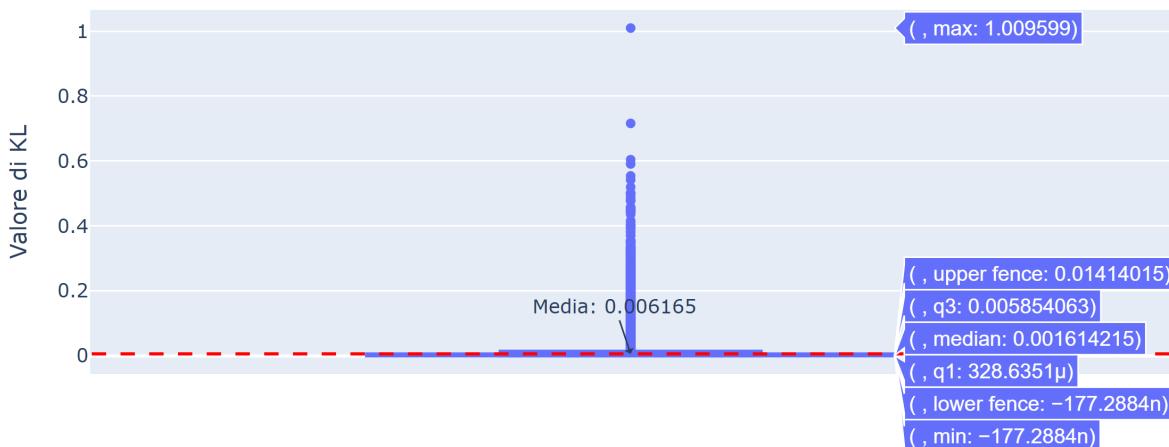


Figura 5.95: Whisker plot |configurazione B, $M = 3$, input = 354294|

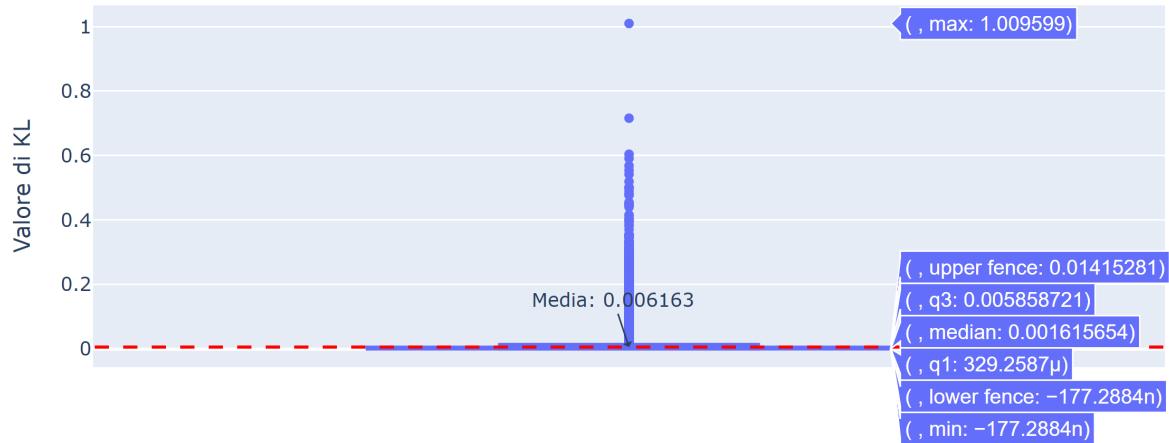


Figura 5.96: Whisker plot |configurazione B, $M = 3$, input = 472392|

Distribuzione delle frequenze con la configurazione C e $M = 3$

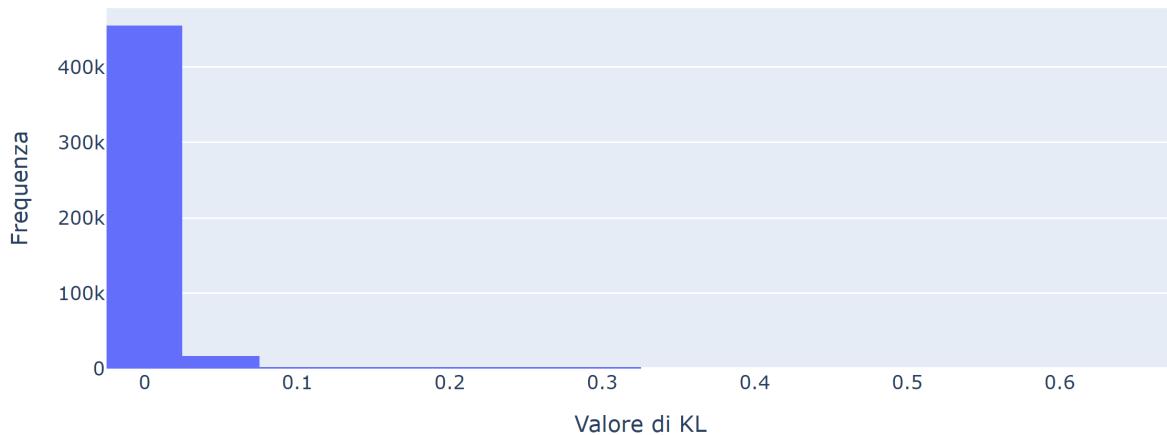


Figura 5.97: Grafici distribuzione |configurazione C, $M = 3$, input = 23620|

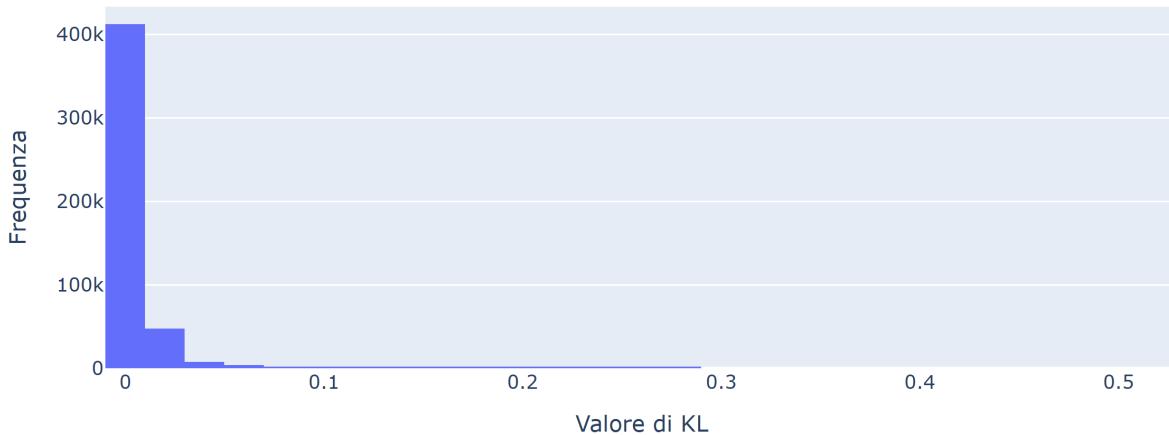


Figura 5.98: Grafici distribuzione |configurazione C, $M = 3$, input = 47239|

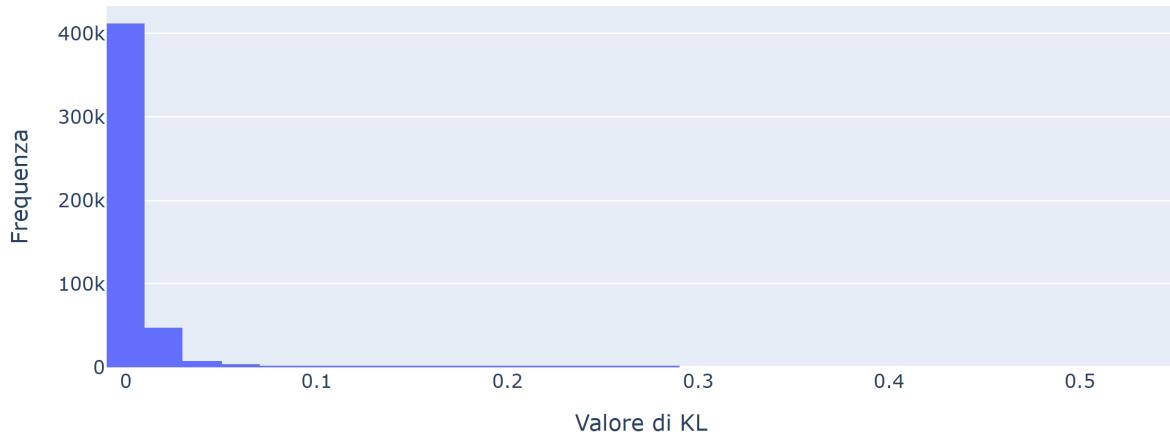


Figura 5.99: Grafici distribuzione |configurazione C, $M = 3$, input = 118098|

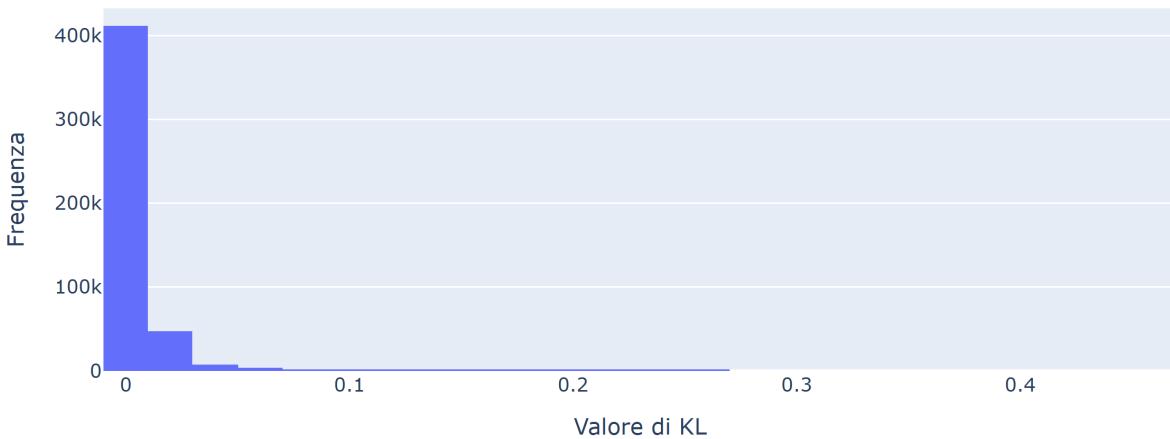


Figura 5.100: Grafici distribuzione |configurazione C, $M = 3$, input = 236196|

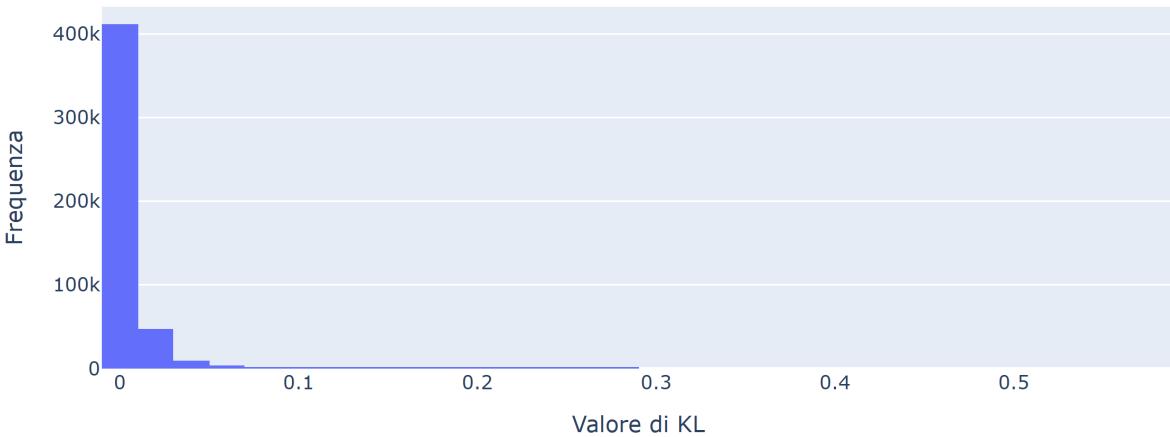


Figura 5.101: Grafici distribuzione |configurazione C, $M = 3$, input = 354294|

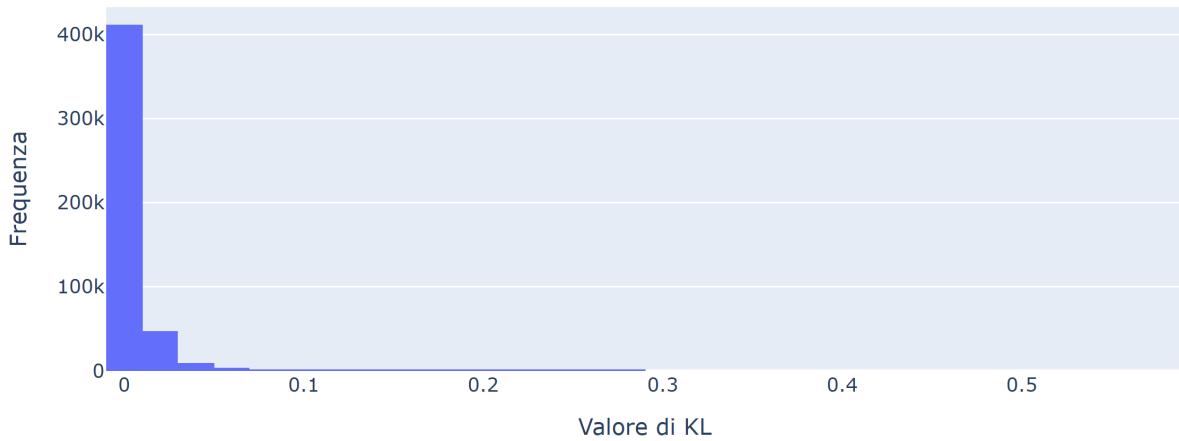


Figura 5.102: Grafici distribuzione |configurazione C, $M = 3$, input = 472392|

Whisker Plot con la configurazione C e $M = 3$

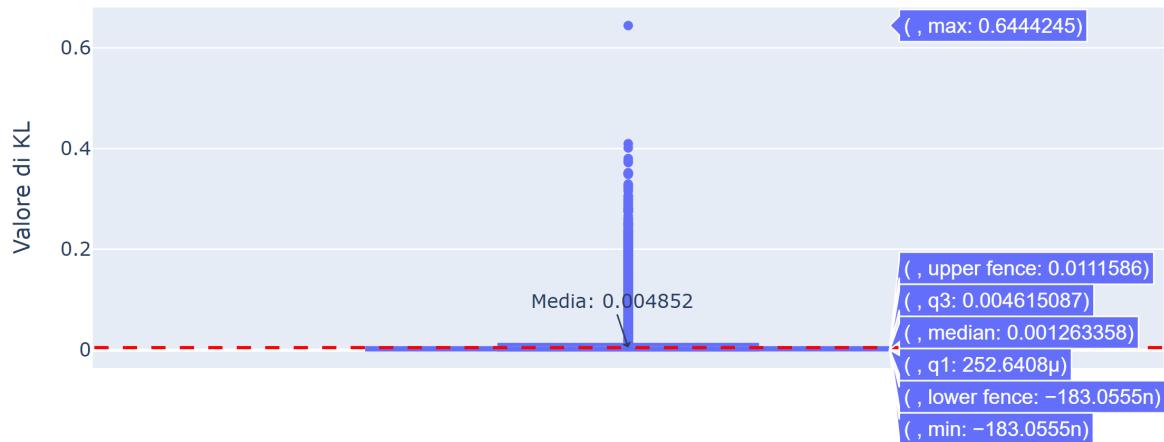


Figura 5.103: Whisker plot |configurazione C, $M = 3$, input = 23620|

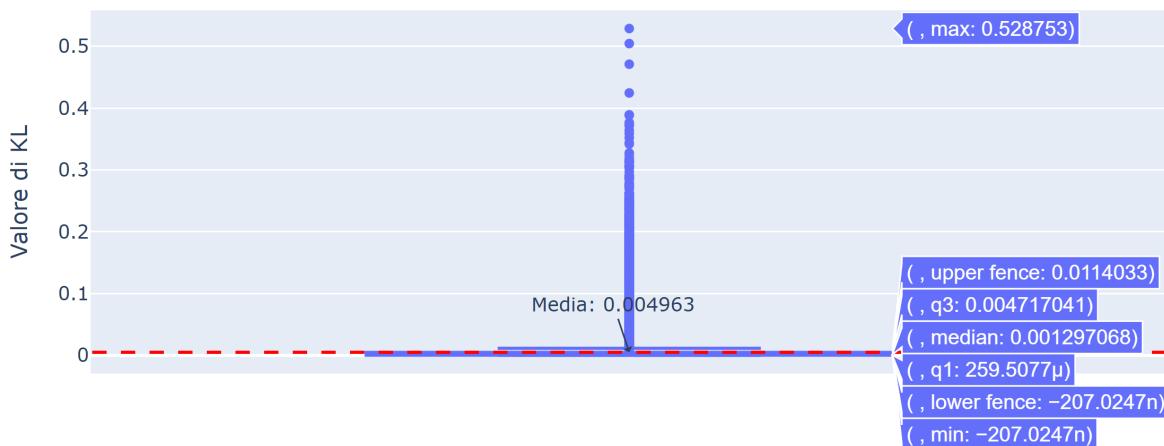


Figura 5.104: Whisker plot |configurazione C, $M = 3$, input = 47239|

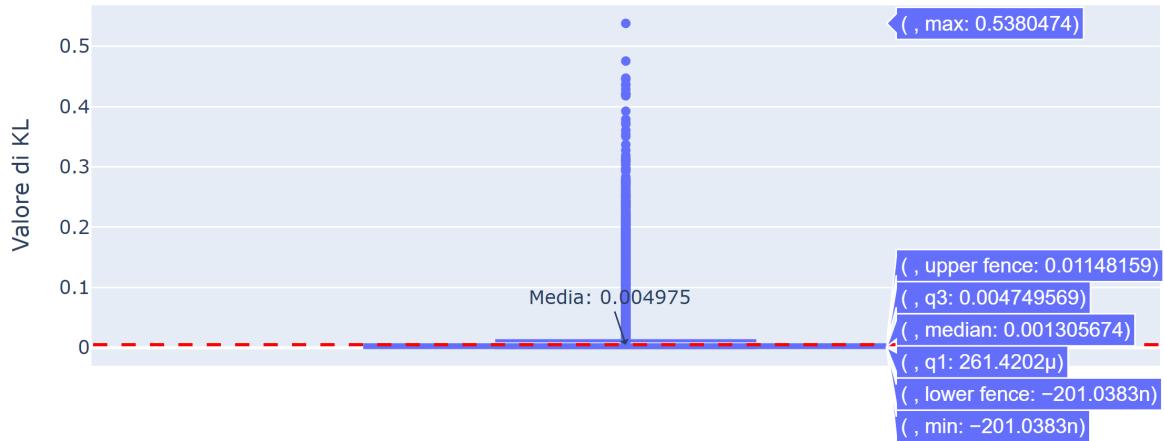


Figura 5.105: Whisker plot |configurazione C, $M = 3$, input = 118098|

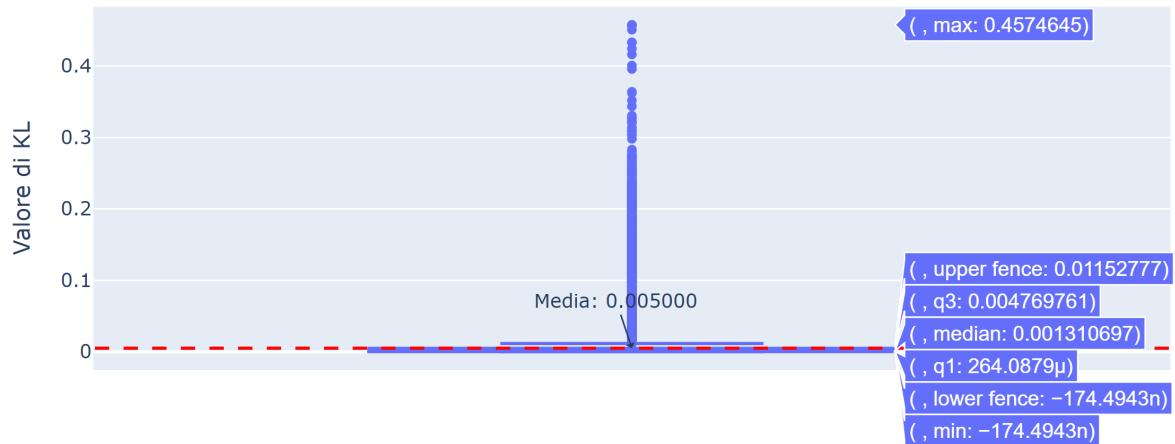


Figura 5.106: Whisker plot |configurazione C, $M = 3$, input = 236196|

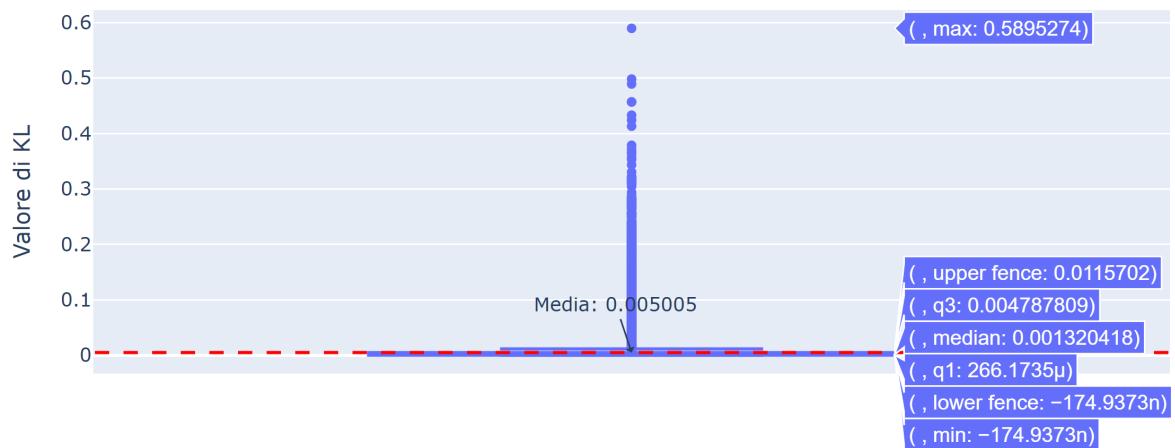


Figura 5.107: Whisker plot |configurazione C, $M = 3$, input = 354294|

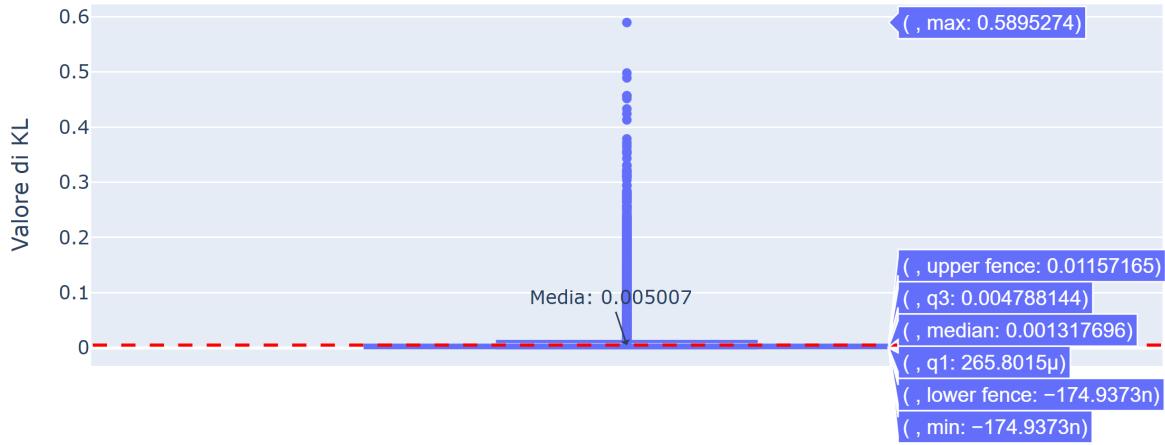


Figura 5.108: Whisker plot |configurazione C, $M = 3$, input = 472392|

5.2.2 Risultati - Quantizzazione statica - Media/Mediana

Descrizione dei Risultati

I risultati riportati nelle seguenti tabelle si riferiscono alle configurazioni che minimizzano la media e la mediana della divergenza di Kullback-Leibler (KL). Queste configurazioni sono state selezionate per analizzare l'impatto della calibrazione sulle performance del modello. Per ciascun valore di M , sono state scelte le configurazioni A, B e C con il numero di input di calibrazione che minimizzano la media e la mediana della divergenza KL. Ad esempio, con $M = 1$, per la media sono state selezionate le configurazioni:

- **A:** Input di calibrazione scelti randomicamente pari al 5% del dataset.
- **B:** Input di calibrazione scelti randomicamente pari al 50% del dataset.
- **C:** Input di calibrazione scelti randomicamente pari al 100% del dataset.

Mentre, per la mediana sono state selezionate le configurazioni:

- **A:** Input di calibrazione scelti randomicamente pari al 10% del dataset.
- **B:** Input di calibrazione scelti randomicamente pari al 5% del dataset.
- **C:** Input di calibrazione scelti randomicamente pari al 5% del dataset.

Per $M = 2$, per la media sono state selezionate le configurazioni:

- **A:** Input di calibrazione scelti randomicamente pari al 5% del dataset.
- **B:** Input di calibrazione scelti randomicamente pari al 10% del dataset.

- **C:** Input di calibrazione scelti randomicamente pari al 5% del dataset.

Mentre, per la mediana sono state selezionate le configurazioni:

- **A:** Input di calibrazione scelti randomicamente pari al 5% del dataset.
- **B:** Input di calibrazione scelti randomicamente pari al 5% del dataset.
- **C:** Input di calibrazione scelti randomicamente pari al 20% del dataset.

Per $M = 2$, le configurazioni selezionate per minimizzare sia la media che la mediana della divergenza di KL sono risultate le stesse, in quanto coincidevano. Le configurazioni scelte sono le seguenti:

- **A:** Input di calibrazione scelti randomicamente pari al 5% del dataset.
- **B:** Input di calibrazione scelti randomicamente pari al 5% del dataset.
- **C:** Input di calibrazione scelti randomicamente pari al 5% del dataset.

KPI	A	B	C	Originale
Goodput	0,4228	0,4224	0,4268	0,4244
Time slots	16,1010	15,9630	15,3785	15,8798
Packet	3,9788	3,9793	3,9810	3,9795
Collision	0,5941	0,6085	0,6030	0,6034
Already transm. packets	0,0119	0,0123	0,0116	0,0113

Tabella 5.9: Risultati Media $M = 1$.

KPI	A	B	C	Originale
Goodput	0,4233	0,4244	0,4227	0,4244
Time slots	15,9731	16,0087	16,1451	15,8798
Packet	3,9792	3,9789	3,9786	3,9795
Collision	0,5950	0,6184	0,6064	0,6034
Already transm. packets	0,0105	0,0136	0,0125	0,0113

Tabella 5.10: Risultati Mediana $M = 1$.

KPI	A	B	C	Originale
Goodput	0,4773	0,4778	0,4733	0,4751
Time slots	13,0508	14,0504	13,7797	13,2681
Packets	3,9848	3,9812	3,9826	3,9842
Collisions	0,5623	0,5664	0,5787	0,5729
Already transmitted packets	0,0656	0,0678	0,0793	0,0739

Tabella 5.11: Risultati Media $M = 2$.

KPI	A	B	C	Originale
Goodput	0,4773	0,4760	0,4763	0,4751
Time slots	13,0508	13,7316	13,2907	13,2681
Packets	3,9848	3,9825	3,9841	3,9842
Collisions	0,5623	0,5540	0,5819	0,5729
Already transmitted packets	0,0656	0,0697	0,0683	0,0739

Tabella 5.12: Risultati Mediana $M = 2$.

KPI	A	B	C	Originale
Goodput	0,4366	0,4372	0,4368	0,4370
Time slots	21,7765	21,5050	20,8243	21,5647
Packets	3,9551	3,9562	3,9585	3,9559
Collisions	0,5664	0,5956	0,6090	0,5911
Already transmitted packets	0,1327	0,1182	0,1087	0,1013

Tabella 5.13: Risultati Media/Mediana $M = 3$.

Anche in questo caso, come è possibile notare dai risultati esposti dalle tabelle precedenti, tutte le performance ottenute dai modelli quantizzati sono in linea con quelle del modello originale.

Per $M = 1$, tra le configurazioni che riducono media e mediana della divergenza KL, quelle che ottengono le performance migliori sono:

- Media: configurazione C con 81 input (50% del dataset);
- Mediana: configurazione A con 16 input (10% del dataset).

Tra le due, la configurazione che riduce la media risulta migliore in 3 casi su 5.

Per $M = 2$, tra le configurazioni che riducono media e mediana della divergenza KL, quelle che ottengono le performance migliori sono:

- Media: configurazione A con 437 input (5% del dataset);
- Mediana: configurazione A con 437 input (5% del dataset).

Praticamente, con $M = 2$, utilizzando la configurazione A con un numero di input pari al 5% del dataset si ottiene la configurazione che performa meglio considerando sia la media sia la mediana.

Per $M = 3$, ogni configurazione ha ridotto media e mediana utilizzando lo stesso numero di input. La configurazione migliore tra le tre è la configurazione C con 23.620 input (5% del dataset).

5.2.3 Risultati - Quantizzazione statica

Sulla base dei risultati ottenuti nel paragrafo precedente, si è deciso di effettuare per ogni valore di M delle simulazioni con la configurazione A utilizzando un numero di input di calibrazione pari al 5% della dimensione totale del dataset. I risultati ottenuti sono esposti nelle tabelle seguenti.

KPI	Int8	Original
Goodput	0,4216	0,4244
Time slots	16,0640	15,8798
Packet	3,9790	3,9795
Collision	0,6009	0,6034
Already transm.packets	0,0121	0,0113

Tabella 5.14: Risultati - Quantizzazione statica $M = 1$.

KPI	Int8	Original
Goodput	0,4773	0,4751
Time slots	13,0508	13,2681
Packet	3,9848	3,9842
Collision	0,5623	0,5729
Already transm.packets	0,0656	0,0739

Tabella 5.15: Risultati - Quantizzazione statica $M = 2$.

KPI	Int8	Original
Goodput	0,4366	0,4370
Time slots	21,7765	21,5647
Packet	3,9551	3,9559
Collision	0,5664	0,5911
Already transm. packets	0,1327	0,1013

Tabella 5.16: Risultati - Quantizzazione statica $M = 3$.

KPI	Int8	Original
Goodput	0,4204	0,4208
Time slots	26,1761	23,9845
Packet	3,9408	3,9848
Collision	0,4439	0,4574
Already transm.packets	0,3308	0,3678

Tabella 5.17: Risultati - Quantizzazione statica $M = 4$.

Memory (M)	Int8	F32-Only actor
$M = 1$	0,0175	0,0228
$M = 2$	0,0177	0,0238
$M = 3$	0,0180	0,0248
$M = 4$	0,0182	0,0257

Tabella 5.18: Memory usage on disk (Mb) - Quantizzazione statica

Memory (M)	Int8	F32-Only actor
$M = 1$	0,0047	0,0188
$M = 2$	0,0049	0,0197
$M = 3$	0,0051	0,0207
$M = 4$	0,0054	0,0257

Tabella 5.19: Memory usage on running (Mb) - Quantizzazione statica

KPI	Int8	Original
Inference Time	0,0050	0,0049

Tabella 5.20: Risultati Quantizzazione Statica - Tempi medi di inferenza a confronto (ms).

Memoria In fase di esecuzione, il modello quantizzato con interi a 8 bit riduce l'occupazione di memoria del 75% rispetto al modello originale. Per quanto riguarda la memoria occupata su disco, il modello `Int8` riduce lo spazio richiesto del 30%.

Prestazioni Per quanto riguarda le prestazioni, il modello quantizzato mostra performance simili a quelle del modello originale e, in alcuni casi, leggermente migliori, come si può osservare dai risultati. Per quanto riguarda il tempo di inferenza, questo risulta simile, con un leggero vantaggio per il modello originale.

In conclusione, in questo caso la scelta cade sul modello originale in quanto il modello quantizzato da benefici solo in termini di memoria.

5.2.4 Risultati - Quantizzazione statica - Custom Config

In base ai risultati ottenuti, è stata presa la decisione di implementare configurazioni personalizzate, al fine di valutare l'impatto della variazione del numero di livelli di quantizzazione. Le configurazioni sperimentate sono descritte in 4.4.2, mentre i risultati ottenuti sono riportati nelle tabelle seguenti.

KPI	Int8	Original
Goodput	0,4518	0,4244
Time slots	11,0417	15,8798
Packet	3,9943	3,9795
Collision	0,8980	0,6034
Already transm. packets	0,0152	0,0113

Tabella 5.21: Risultati - Quantizzazione statica $M = 1$ - Custom config

KPI	Int8	Original
Goodput	0,4759	0,4751
Time slots	10,5961	13,2681
Packet	3,9934	3,9842
Collision	0,6565	0,5729
Already transm. packets	0,1244	0,0739

Tabella 5.22: Risultati - Quantizzazione statica $M = 2$ - Custom config

KPI	Int8	Original
Goodput	0,4402	0,4370
Time slots	19,6862	21,5647
Packet	3,9621	3,9559
Collision	0,6770	0,5911
Already transm. packets	0,0644	0,1013

Tabella 5.23: Risultati - Quantizzazione statica $M = 3$ - Custom config

KPI	Int8	Original
Goodput	0,4326	0,4208
Time slots	22,5313	23,9845
Packet	3,9530	3,9848
Collision	0,4593	0,4574
Already transm. packets	0,2374	0,3678

Tabella 5.24: Risultati - Quantizzazione statica $M = 4$ - Custom config

Prestazioni Per quanto riguarda le prestazioni, il modello quantizzato ha performance migliori rispetto quelle del modello originale, come è possibile vedere dai risultati.

In conclusione, la modifica dei parametri di quantizzazione altera la rappresentazione numerica dei pesi e delle attivazioni del modello, influenzando la distribuzione interna del modello. Tale alterazione si ripercuote sull'uscita finale, in particolare sulla funzione softmax, modificando così il risultato delle predizioni. In alcuni casi, ciò può portare a miglioramenti senza la necessità di riaddestrare il modello. In pratica, questi risultati suggeriscono che il modello originale aveva delle inefficienze o delle imperfezioni nel training che la quantizzazione ha risolto.

5.3 Risultati - Quantization Aware Training - Default

Lo step del fine-tuning relativo alla QAT è stato effettuato con gli stessi hyperparametri di training del training originale, fatta eccezione per il numero di simulazioni e i learning rate, cui valori sono riportati nella tabella seguente.

Parametro	Simbolo	Valore
Numero di simulazioni	N_{sim}	1000
Learning rate	L_r	$3 * 10^{-5}$
Learning rate critec	L_c	10^{-4}

Tabella 5.25: Parametri fine-tuning QAT - default - $M = \{1, 2, 3, 4\}$.

KPI	Int8	Original
Goodput	0,4709	0,4244
Time slots	15,6742	15,8798
Packet	3.9758	3,9795
Collision	0,6033	0,6034
Already transm. packets	0,0124	0,0113

Tabella 5.26: Risultati - QAT - default - $M = 1$.

KPI	Int8	Original
Goodput	0,4880	0,4751
Time slots	14,4560	13,2681
Packet	3,9788	3,9842
Collision	0,4653	0,5729
Already transm. packets	0,1087	0,0739

Tabella 5.27: Risultati - QAT - default - $M = 2$.

KPI	Int8	Original
Goodput	0,4535	0,4370
Time slots	20,3145	21,5647
Packet	3,9586	3,9559
Collision	0,5320	0,5911
Already transm. packets	0,0881	0,1013

Tabella 5.28: Risultati - QAT - default - $M = 3$.

KPI	Int8	Original
Goodput	0,4516	0,4208
Time slots	23,0969	23,9845
Packet	3,9499	3,9848
Collision	0,5217	0,4574
Already transm. packets	0,2936	0,3678

 Tabella 5.29: Risultati - QAT - default - $M = 4$.

Memory (M)	Int8	F32-Only actor
$M = 1$	0,0164	0,0228
$M = 2$	0,0123	0,0238
$M = 3$	0,0125	0,0248
$M = 4$	0,0128	0,0257

Tabella 5.30: Memory usage on disk (Mb) - QAT.

Memory (M)	Int8	F32-Only actor
$M = 1$	0,0047	0,0188
$M = 2$	0,0049	0,0197
$M = 3$	0,0051	0,0207
$M = 4$	0,0054	0,0217

Tabella 5.31: Memory usage on running (Mb) - QAT.

KPI	Int8	Original
Inference Time	0,0050	0,0049

Tabella 5.32: Risultati QAT - Tempi medi di inferenza a confronto (ms).

Memoria Per quel che concerne la memoria occupata su disco, con il modello quantizzato si risparmia fino al 50% rispetto a quella occupata dal modello originale. Per quanto riguarda la memoria utilizzata in esecuzione, quella richiesta dal modello quantizzato è 1/4 rispetto a quella richiesta dal modello originale.

Prestazioni Per quanto riguarda le prestazioni ottenute con la QAT, come si può notare dalle tabelle precedentemente esposte, la maggior parte delle KPI misurate è a

favore del modello originale, tranne per $M = 3$; mentre, per quanto concerne i tempi di inferenza, quello del modello originale è leggermente inferiore.

In conclusione, non conviene utilizzare la tecnica di quantizzazione di tipo QAT con parametri di default, in quanto i benefici che si ottengono non giustificano la consistente perdita di performance.

5.4 Risultati - Quantization Aware Training - Custom

Come per la quantizzazione statica anche qui è stata adoperata una configurazione di quantizzazione personalizzate per vedere se le performance miglioravano, questa configurazione è descritta nel paragrafo 4.4.3. Inoltre, sono stati modificati alcuni hyperparametri di training, cui valori sono riportati nelle tabelle seguenti, per rendere più efficace il finetuning.

KPI	Int8	Original
Goodput	0,4807	0,4244
Time slots	11,1437	15,8798
Packet	3.9908	3,9795
Collision	0,6894	0,6034
Already transm. packets	0,0032	0,0113

Tabella 5.33: Risultati - QAT $M = 1$.

KPI	Int8	Original
Goodput	0,4983	0,4751
Time slots	10,2625	13,2681
Packet	3,9929	3,9842
Collision	0,4791	0,5729
Already transm.packets	0,0487	0,0739

Tabella 5.34: Risultati - QAT $M = 2$.

KPI	Int8	Original
Goodput	0,4533	0,4370
Time slots	18,2555	21,5647
Packet	3,9660	3,9559
Collision	0,4822	0,5911
Already transm. packets	0,0958	0,1013

Tabella 5.35: Risultati - QAT $M = 3$.

KPI	Int8	Original
Goodput	0,4714	0,4208
Time slots	18,4065	23,9845
Packet	3,9648	3,9848
Collision	0,5226	0,4574
Already transm. packets	0,2936	0,3678

Tabella 5.36: Risultati - QAT $M = 4$.

Prestazioni Per quanto riguarda le prestazioni ottenute con la QAT, come si può notare dalle tabelle precedentemente esposte, la maggior parte delle KPI misurate è a favore del modello quantizzato; mentre, per quanto concerne i tempi di inferenza, quello del modello originale è leggermente inferiore.

In conclusione, alla luce dei risultati ottenuti, anche se il modello quantizzato è leggermente più lento, esso è preferibile poiché migliora significativamente anche le prestazioni. La ragione di ciò potrebbe essere che la modifica delle probabilità abbia portato a un protocollo ancora migliore di quello di partenza, il quale potrebbe essere incappato in un minimo locale. Pertanto, se applicata in modo avanzato, la QAT può migliorare non solo le prestazioni, ma anche l'uso della memoria.

5.5 Risultati - Confronto fra i diversi tipi di quantizzazione

KPI	Dyn.Int8	Dyn.Float16	Static	QAT-Default	QAT-Custom	Original
Goodput	0,4231	0,4238	0,4518	0,4709	0,4807	0,4244
Time slots	15,5364	15,9588	11,0417	15,6742	11,1437	15,8798
Packet	3,9808	3,9792	3,9943	3,9758	3,9908	3,97795
Collision	0,6024	0,6057	0,8980	0,6033	0,6894	0,6034
Already transm. packets	0,0126	0,0116	0,0152	0,0124	0,0032	0,0113

Tabella 5.37: Risultati a confronto $M = 1$

KPI	Dyn.Int8	Dyn.Float16	Static	QAT-Default	QAT-Custom	Original
Goodput	0,4782	0,4761	0,4759	0,4880	0,4983	0,4751
Time slots	13,8639	13,0698	10,5961	14,4560	10,2625	13,2681
Packet	3,9819	3,9848	3,9934	3,9788	3,9929	3,9842
Collision	0,5600	0,5683	0,6565	0,4653	0,4791	0,5729
Already transm. packets	0,0674	0,0727	0,1244	0,1087	0,0487	0,0739

Tabella 5.38: Risultati a confronto $M = 2$

KPI	Dyn.Int8	Dyn.Float16	Static	QAT-Default	QAT-Custom	Original
Goodput	0,4370	0,4371	0,4402	0,4535	0,4533	0,4370
Time slots	21,6376	24,4247	19,6862	20,3145	18,2555	21,5647
Packet	3,9555	3,9563	3,9621	3,9586	3,9660	3,9559
Collision	0,5892	0,5907	0,6770	0,5320	0,4822	0,5911
Already transm. packets	0,1228	0,1018	0,0644	0,0881	0,0958	0,1013

Tabella 5.39: Risultati a confronto $M = 3$

KPI	Dyn.Int8	Dyn.Float16	Static	QAT-Default	QAT-Custom	Original
Goodput	0,4222	0,4210	0,4326	0,4516	0,4714	0,4208
Time slots	23,1216	24,1642	22,5313	23,0969	18,4065	23,9845
Packet	3,9517	3,9480	3,9530	3,9499	3,9648	3,9848
Collision	0,4453	0,4566	0,4593	0,5217	0,5226	0,4574
Already transm. packets	0,3718	0,3703	0,2374	0,2936	0,2936	0,3678

Tabella 5.40: Risultati a confronto $M = 4$

Memory (M)	Dyn. Int8	Dyn. Float16	Static	QAT	Original
$M = 1$	0,0121	0,0257	0,0175	0,0164	0,0228
$M = 2$	0,0123	0,0267	0,0177	0,0123	0,0238
$M = 3$	0,0126	0,0276	0,0180	0,0125	0,0248
$M = 4$	0,0128	0,0286	0,0182	0,0128	0,0257

Tabella 5.41: Confronto - Memory usage on disk (Mb).

Memory (M)	Dyn. Int8	Dyn. Float16	Static	QAT	Original
$M = 1$	0,0047	0,0094	0,0047	0,0047	0,0188
$M = 2$	0,0049	0,0098	0,0049	0,0049	0,0197
$M = 3$	0,0051	0,0103	0,0051	0,0051	0,0207
$M = 4$	0,0054	0,0108	0,0054	0,054	0,0217

Tabella 5.42: Confronto - Memory usage while running (Mb).

KPI	dyn. Int8	Float16	static Int8	QAT	Original
Inference Time	0,0045	0,0043	0,0050	0,0050	0,0049

Tabella 5.43: Risultati Complessivi - Tempi medi di inferenza a confronto.

Memoria Per quanto riguarda la memoria occupata su disco, i tipi di quantizzazione che la riducono maggiormente sono la QAT e la quantizzazione dinamica. Invece, per la memoria utilizzata durante l'esecuzione, tutti i tipi di quantizzazione Int8 richiedono la stessa quantità di memoria.

Prestazioni Per quanto concerne le prestazioni, come previsto, la QAT risulta essere la soluzione con le migliori performance, seguita dalla quantizzazione statica. Se però non è possibile effettuare una fase di training, allora la scelta dipende dalle priorità: se si vuole preservare le performance, accettando un maggiore consumo di memoria e tempi di inferenza più lunghi, si opta per la quantizzazione statica; se invece l'obiettivo è ridurre al minimo i tempi di inferenza e lo spazio su disco, anche a costo di un calo delle prestazioni, allora la quantizzazione dinamica è la soluzione migliore.

Capitolo 6

Conclusioni

Questa tesi si è concentrata sull’implementazione di una soluzione basata sul paradigma Multi-agent Reinforcement Learning per l’apprendimento di protocolli MAC adatti a un ambiente industriale e sull’ottimizzazione delle Deep Neural Network tramite tecniche di quantizzazione. L’obiettivo è rendere la soluzione sviluppata più leggera e adatta all’esecuzione su dispositivi IoT, senza compromettere troppo le prestazioni.

Il confronto tra le diverse tecniche di quantizzazione ha mostrato che, per quanto riguarda la memoria utilizzata in esecuzione, tutti i tipi di quantizzazione riducono del 75% la memoria utilizzata. Per quanto riguarda la memoria occupata su disco, la quantizzazione dinamica occupa fino al 50% in meno, la quantizzazione statica fino al 30% in meno e la QAT fino al 50% in meno.

Per quanto riguarda le prestazioni ottenute, con la quantizzazione dinamica si sono ottenute performance in linea con la soluzione proposta, con un minimo calo di precisione e con un riduzione di circa il 10% del tempo di inferenza rispetto il tempo di inferenza del modello originale.

Con la quantizzazione statica, con la configurazione A si sono ottenuti dei parametri in linea con il modello originale; mentre, con la configurazione personalizzata si è assistito a un miglioramento di alcuni KPI, come goodput e tempo di slot, e a un peggioramento di altri KPI, come il numero di collisioni. Per la quantizzazione statica il tempo di inferenza ha un incremento di circa il 2% rispetto il tempo di inferenza del modello originale.

Infine, per quanto riguarda la QAT, con la prima configurazione la maggior parte dei valori delle KPI rimane in linea con i valori del modello originale. Mentre con la configurazione personalizzata si è osservato un miglioramento delle performance nella maggior parte dei KPI. Per quel che concerne il tempo di inferenza, si assiste ad un incremento di circa il 4% rispetto al tempo di inferenza del modello originale.

In generale, la quantizzazione che ha dato le performance migliori è stata la QAT per quanto riguarda i KPI di comunicazione, mentre, per quel che concerne il tempo di inferenza, la quantizzazione che lo riduce maggiormente è quella dinamica; su questo argomento, inoltre e bene ricordare che per modelli poco complessi come quello preso in considerazione, le operazioni di quantizzazione e dequantizzazione introducono un overhead che vanifica il vantaggio ottenuto dall'utilizzo dei pesi interi.

In conclusione, la quantizzazione è una tecnica che permette di ridurre la memoria occupata sia in esecuzione che sul dispositivo di storage, senza compromettere troppo le prestazioni e, in alcuni casi, migliorandole. La tecnica di quantizzazione da utilizzare nel caso di applicazione nel contesti dei MAC protocol dipende dal tempo a disposizione: se si ha poco tempo, la soluzione migliore è quella dinamica, seguita dalla statica e, infine, dalla QAT. Se invece il tempo di deployment è abbastanza lungo, si consiglia di applicare la QAT, in quanto consente di ottenere performance più vicine al modello originale o addirittura superiori, se eseguita con una buona configurazione e un buon fine-tuning; tuttavia, il fine-tuning aumenta notevolmente la complessità di implementazione. Se questa difficoltà rappresenta un problema, si consiglia di usare la quantizzazione statica, che è più semplice da applicare, poiché richiede la configurazione di meno parametri. Se anche questa risulta difficile da applicare, allora l'opzione è quella di utilizzare la quantizzazione dinamica.

Invece, se si considerano le prestazioni dei modelli quantizzati nella loro configurazione predefinita, senza interventi di ottimizzazione o configurazioni avanzate, la quantizzazione statica risulta essere la più precisa, seguita dalla quantizzazione dinamica. Per quel che concerne la QAT, la valutazione delle prestazioni risulta più complessa, in quanto questa tecnica richiede una configurazione più avanzata e un processo di addestramento specifico. Tuttavia, se si considerano le prestazioni in un contesto ottimizzato, la QAT si dimostra la soluzione più efficace, seguita dalla quantizzazione statica. Un possibile lavoro futuro potrebbe includere un'analisi sul perché il tempo di inferenza ottenuto con la quantizzazione dinamica è inferiore rispetto a quello ottenuto con la quantizzazione statica, in quanto in teoria non dovrebbe essere così.

Acknowledgment



This work has been partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on “Telecommunications of the Future” (PE00000001 - program “RESTART”), structural project S9 - Industrial Networks (IN).

Indice delle Figure

1.1	Evoluzione delle reti radio mobili [8].	3
1.2	Confronto tra 6G e 5G [10].	4
1.3	Elenco delle capacità delle rete 6G a confronto con quelle del 5G [9].	5
1.4	Illustrazione degli scenari d'uso di IMT-2030 e confronto con IMT-2020 [9].	7
1.5	Tony Stark alle prese col modello digitale della sua armatura [12].	8
1.6	Riproduzione di un messaggio olografico in un film della saga di <i>Star Wars</i> [13].	9
1.7	Un robot corriere intelligente [14].	10
1.8	Architettura della rete 5G [15].	11
1.9	NFV architectural framework [5].	11
1.10	SDN basic architecture [6].	12
1.11	SBA architectural [7].	12
1.12	Esempio di slicing network [4].	13
2.1	Esempio reale di Reinforcement Learning [17].	19
2.2	Schema di funzionamento del Reinforcement Learning [19].	21
2.3	Classificazione degli algoritmi di Reinforcement Learning [20].	21
2.4	Ambiente single agent [25].	25
2.5	Ambiente multi-agent [25].	26
2.6	Esempio MARL [26].	26
2.7	Esempio FCL [28].	29
2.8	Esempio IL [28].	31
2.9	Esempio CTDE [28].	32
2.10	CTDE con struttura Actor-Critic [29].	32
3.1	Esempio di quantizzazione simmetrica con intervallo quantizzato uint8.	38
3.2	Esempio di quantizzazione simmetrica con intervallo quantizzato int8.	38

3.3	Esempio di quantizzazione simmetrica con perdita di intervallo utile [-127, -1].	39
3.4	Esempio di quantizzazione simmetrica con perdita di valori rappresentabili [-2, 0].	39
3.5	Esempio di quantizzazione asimmetrica con intervallo quantizzato in uint8. .	40
3.6	Esempio di quantizzazione asimmetrica con intervallo quantizzato in uint8. .	40
3.7	Un'immagine quantizzata confrontata con l'originale [34].	42
3.8	Esempio sul calcolo dell'errore di quantizzazione [34].	43
3.9	Errori di quantizzazione ottenuti con Δ_1 e Δ_2	46
3.10	Frequenza errori di quantizzazione con Δ_1 e Δ_2	47
3.11	Step principali della quantizzazione post training statica [40].	51
3.12	Esempio di applicazione dei FakeQuant [43].	52
3.13	Illustrazione della QAT durante la forward e la backpropagation [42]. .	53
3.14	Step principali della quantizzazione aware training [40].	54
4.1	Rappresentazione ad alto livello del modello. [45]	57
4.2	Struttura del modello DNN implementato [45].	62
4.3	Grafico della funzione di attivazione tanh [47].	63
4.4	Illustrazione del funzionamento della softmax [48].	63
5.1	Grafici distribuzione configurazione A, $M = 1$, input = 8 	81
5.2	Grafici distribuzione configurazione A, $M = 1$, input = 16 	82
5.3	Grafici distribuzione configurazione A, $M = 1$, input = 40 	82
5.4	Grafici distribuzione configurazione A, $M = 1$, input = 81 	82
5.5	Grafici distribuzione configurazione A, $M = 1$, input = 122 	83
5.6	Grafici distribuzione configurazione A, $M = 1$, input = 162 	83
5.7	Whisker plot configurazione A, $M = 1$, input = 8 	83
5.8	Whisker plot configurazione A, $M = 1$, input = 16 	84
5.9	Whisker plot configurazione A, $M = 1$, input = 40 	84
5.10	Whisker plot configurazione A, $M = 1$, input = 81 	84
5.11	Whisker plot configurazione A, $M = 1$, input = 122 	85
5.12	Whisker plot configurazione A, $M = 1$, input = 162 	85
5.13	Grafici distribuzione configurazione B, $M = 1$, input = 8 	85
5.14	Grafici distribuzione configurazione B, $M = 1$, input = 16 	86
5.15	Grafici distribuzione configurazione B, $M = 1$, input = 40 	86
5.16	Grafici distribuzione configurazione B, $M = 1$, input = 81 	86
5.17	Grafici distribuzione configurazione B, $M = 1$, input = 122 	87

5.18	Grafici distribuzione configurazione B, $M = 1$, input = 162 	87
5.19	Whisker plot configurazione B, $M = 1$, input = 8 	87
5.20	Whisker plot configurazione B, $M = 1$, input = 16 	88
5.21	Whisker plot configurazione B, $M = 1$, input = 40 	88
5.22	Whisker plot configurazione B, $M = 1$, input = 81 	88
5.23	Whisker plot configurazione B, $M = 1$, input = 122 	89
5.24	Whisker plot configurazione B, $M = 1$, input = 162 	89
5.25	Grafici distribuzione configurazione C, $M = 1$, input = 8 	89
5.26	Grafici distribuzione configurazione C, $M = 1$, input = 16 	90
5.27	Grafici distribuzione configurazione C, $M = 1$, input = 40 	90
5.28	Grafici distribuzione configurazione C, $M = 1$, input = 81 	90
5.29	Grafici distribuzione configurazione C, $M = 1$, input = 122 	91
5.30	Grafici distribuzione configurazione C, $M = 1$, input = 162 	91
5.31	Whisker plot configurazione C, $M = 1$, input = 8 	91
5.32	Whisker plot configurazione C, $M = 1$, input = 16 	92
5.33	Whisker plot configurazione C, $M = 1$, input = 40 	92
5.34	Whisker plot configurazione C, $M = 1$, input = 81 	92
5.35	Whisker plot configurazione C, $M = 1$, input = 122 	93
5.36	Whisker plot configurazione C, $M = 1$, input = 162 	93
5.37	Whisker plot configurazione A, $M = 2$, input = 437 	93
5.38	Whisker plot configurazione A, $M = 2$, input = 875 	94
5.39	Whisker plot configurazione A, $M = 2$, input = 2187 	94
5.40	Whisker plot configurazione A, $M = 2$, input = 4374 	94
5.41	Whisker plot configurazione A, $M = 2$, input = 6561 	95
5.42	Whisker plot configurazione A, $M = 2$, input = 8748 	95
5.43	Whisker plot configurazione A, $M = 2$, input = 437 	95
5.44	Whisker plot configurazione A, $M = 2$, input = 875 	96
5.45	Whisker plot configurazione A, $M = 2$, input = 2187 	96
5.46	Whisker plot configurazione A, $M = 2$, input = 4374 	96
5.47	Whisker plot configurazione A, $M = 2$, input = 6561 	97
5.48	Whisker plot configurazione A, $M = 2$, input = 8748 	97
5.49	Grafici distribuzione configurazione B, $M = 2$, input = 437 	97
5.50	Grafici distribuzione configurazione B, $M = 2$, input = 875 	98
5.51	Grafici distribuzione configurazione B, $M = 2$, input = 2187 	98
5.52	Grafici distribuzione configurazione B, $M = 2$, input = 4374 	98
5.53	Grafici distribuzione configurazione B, $M = 2$, input = 6561 	99

5.54	Grafici distribuzione configurazione B, $M = 2$, input = 8748 	99
5.55	Whisker plot configurazione B, $M = 2$, input = 437 	99
5.56	Whisker plot configurazione B, $M = 2$, input = 875 	100
5.57	Whisker plot configurazione B, $M = 2$, input = 2187 	100
5.58	Whisker plot configurazione B, $M = 2$, input = 4374 	100
5.59	Whisker plot configurazione B, $M = 2$, input = 6561 	101
5.60	Whisker plot configurazione B, $M = 2$, input = 8748 	101
5.61	Grafici distribuzione configurazione C, $M = 2$, input = 437 	101
5.62	Grafici distribuzione configurazione C, $M = 2$, input = 875 	102
5.63	Grafici distribuzione configurazione C, $M = 2$, input = 2187 	102
5.64	Grafici distribuzione configurazione C, $M = 2$, input = 4374 	102
5.65	Grafici distribuzione configurazione C, $M = 2$, input = 6561 	103
5.66	Grafici distribuzione configurazione C, $M = 2$, input = 8748 	103
5.67	Whisker plot configurazione C, $M = 2$, input = 437 	103
5.68	Whisker plot configurazione C, $M = 2$, input = 875 	104
5.69	Whisker plot configurazione C, $M = 2$, input = 2187 	104
5.70	Whisker plot configurazione C, $M = 2$, input = 4374 	104
5.71	Whisker plot configurazione C, $M = 2$, input = 6561 	105
5.72	Grafici distribuzione configurazione C, $M = 2$, input = 23620 	105
5.73	Grafici distribuzione configurazione A, $M = 3$, input = 23620 	105
5.74	Grafici distribuzione configurazione A, $M = 3$, input = 47239 	106
5.75	Grafici distribuzione configurazione A, $M = 3$, input = 118098 	106
5.76	Grafici distribuzione configurazione A, $M = 3$, input = 236196 	106
5.77	Grafici distribuzione configurazione A, $M = 3$, input = 354294 	107
5.78	Grafici distribuzione configurazione A, $M = 3$, input = 472392 	107
5.79	Whisker plot configurazione A, $M = 3$, input = 23620 	107
5.80	Whisker plot configurazione A, $M = 3$, input = 47239 	108
5.81	Whisker plot configurazione A, $M = 3$, input = 118098 	108
5.82	Whisker plot configurazione A, $M = 3$, input = 236196 	108
5.83	Whisker plot configurazione A, $M = 3$, input = 354294 	109
5.84	Whisker plot configurazione A, $M = 3$, input = 472392 	109
5.85	Grafici distribuzione configurazione B, $M = 3$, input = 23620 	109
5.86	Grafici distribuzione configurazione B, $M = 3$, input = 47239 	110
5.87	Grafici distribuzione configurazione B, $M = 3$, input = 118098 	110
5.88	Grafici distribuzione configurazione B, $M = 3$, input = 236196 	110
5.89	Grafici distribuzione configurazione B, $M = 3$, input = 354294 	111

5.90 Grafici distribuzione configurazione B, $M = 3$, input = 472392 	111
5.91 Whisker plot configurazione B, $M = 3$, input = 23260 	112
5.92 Whisker plot configurazione B, $M = 3$, input = 47239 	112
5.93 Whisker plot configurazione B, $M = 3$, input = 118098 	113
5.94 Whisker plot configurazione B, $M = 3$, input = 236196 	113
5.95 Whisker plot configurazione B, $M = 3$, input = 354294 	113
5.96 Whisker plot configurazione B, $M = 3$, input = 472392 	114
5.97 Grafici distribuzione configurazione C, $M = 3$, input = 23620 	114
5.98 Grafici distribuzione configurazione C, $M = 3$, input = 47239 	114
5.99 Grafici distribuzione configurazione C, $M = 3$, input = 118098 	115
5.100Grafici distribuzione configurazione C, $M = 3$, input = 236196 	115
5.101Grafici distribuzione configurazione C, $M = 3$, input = 354294 	115
5.102Grafici distribuzione configurazione C, $M = 3$, input = 472392 	116
5.103Whisker plot configurazione C, $M = 3$, input = 23620 	116
5.104Whisker plot configurazione C, $M = 3$, input = 47239 	116
5.105Whisker plot configurazione C, $M = 3$, input = 118098 	117
5.106Whisker plot configurazione C, $M = 3$, input = 236196 	117
5.107Whisker plot configurazione C, $M = 3$, input = 354294 	117
5.108Whisker plot configurazione C, $M = 3$, input = 472392 	118

Indice dei Codici

3.1	Codice python esempio quantizzazione - parte 1	45
4.1	Codice python rappresentate la struttura del modello.	62
4.2	Codice python - Misurazione del tempo di inferenza.	65
4.1	Codice python - Esempio di quantizzazione dinamica.	67
4.1	Codice python - Prima configurazione.	67
4.1	Codice python - Seconda configurazione.	68
4.1	Codice python - Configurazione quantizzazione statica	68
4.1	Codice python - Configurazione custom $M = 1$	68
4.1	Codice python - Configurazione custom $M = 2$	69
4.1	Codice python - Configurazione custom $M = 3, 4$	69
4.1	Codice python della calibrazione	71
4.1	Codice python divergenza KL	72
4.2	Codice python creazione dataset	73
4.1	Codice python della configurazione QAT	74
4.1	Codice python rappresentante la prepare e la policy adottata	75
4.1	Codice python rappresentante la convert e il salvataggio del modello addestrato	75

Indice delle Tabelle

4.1	Parametri del training	61
4.2	Parametri fine-tuning - Configurazione A - $M = \{1, 2, 3, 4\}$	76
4.3	Parametri fine-tuning - Custom - $M = \{1, 2\}$	76
4.4	Parametri fine-tuning - Custom - $M = 3$	76
4.5	Parametri fine-tuning - Custom - $M = 4$	77
5.1	Parametri della simulazione - quantizzazione dinamica	78
5.2	Risultati - Quantizzazione dinamica $M = 1$	79
5.3	Risultati - Quantizzazione dinamica $M = 2$	79
5.4	Risultati - Quantizzazione dinamica $M = 3$	79
5.5	Risultati - Quantizzazione dinamica $M = 4$	79
5.6	Memory usage on disk (Mb) - Quantizzazione dinamica	80
5.7	Memory usage while running (Mb) - Quantizzazione dinamica	80
5.8	Risultati Quantizzazione Dinamica - Tempi medi di inferenza a confronto.	80
5.9	Risultati Media $M = 1$	119
5.10	Risultati Mediana $M = 1$	119
5.11	Risultati Media $M = 2$	120
5.12	Risultati Mediana $M = 2$	120
5.13	Risultati Media/Mediana $M = 3$	120
5.14	Risultati - Quantizzazione statica $M = 1$	121
5.15	Risultati - Quantizzazione statica $M = 2$	121
5.16	Risultati - Quantizzazione statica $M = 3$	122
5.17	Risultati - Quantizzazione statica $M = 4$	122
5.18	Memory usage on disk (Mb) - Quantizzazione statica	122
5.19	Memory usage on running (Mb) - Quantizzazione statica	122
5.20	Risultati Quantizzazione Statica - Tempi medi di inferenza a confronto (ms).	122
5.21	Risultati - Quantizzazione statica $M = 1$ - Custom config	123

5.22 Risultati - Quantizzazione statica $M = 2$ - Custom config	123
5.23 Risultati - Quantizzazione statica $M = 3$ - Custom config	124
5.24 Risultati - Quantizzazione statica $M = 4$ - Custom config	124
5.25 Parametri fine-tuning QAT - default - $M = \{1, 2, 3, 4\}$	125
5.26 Risultati - QAT - default - $M = 1$	125
5.27 Risultati - QAT - default - $M = 2$	125
5.28 Risultati - QAT - default - $M = 3$	125
5.29 Risultati - QAT - default - $M = 4$	126
5.30 Memory usage on disk (Mb) - QAT.	126
5.31 Memory usage on running (Mb) - QAT.	126
5.32 Risultati QAT - Tempi medi di inferenza a confronto (ms).	126
5.33 Risultati - QAT $M = 1$	127
5.34 Risultati - QAT $M = 2$	127
5.35 Risultati - QAT $M = 3$	128
5.36 Risultati - QAT $M = 4$	128
5.37 Risultati a confronto $M = 1$	129
5.38 Risultati a confronto $M = 2$	129
5.39 Risultati a confronto $M = 3$	129
5.40 Risultati a confronto $M = 4$	129
5.41 Confronto - Memory usage on disk (Mb).	130
5.42 Confronto - Memory usage while running (Mb).	130
5.43 Risultati Complessivi - Tempi medi di inferenza a confronto.	130

Bibliografia

- [1] Christopher Cox. *An Introduction to LTE*. Wiley, 2012.
- [2] C.-X. Wang, X. You, X. Gao e et al. «On the road to 6G: Visions, requirements, key technologies, and testbeds». In: *IEEE Communications Surveys & Tutorials* 25.2 (2023), pp. 905–974. DOI: 10.1109/COMST.2023.3249835.
- [3] X. You, C. Wang, J. Huang e et al. «Towards 6G wireless communication networks: Vision, enabling technologies, and new paradigm shifts». In: *Science China Information Sciences* 64 (2021). Received 17 May 2020, Revised 08 June 2020, Accepted 17 June 2020, Published 24 November 2020, p. 110301. DOI: 10.1007/s11432-020-2955-6. URL: <https://doi.org/10.1007/s11432-020-2955-6>.
- [4] Afra Domeke, Bruno Cimoli e Idelfonso Tafur Monroy. «Integration of Network Slicing and Machine Learning into Edge Networks for Low-Latency Services in 5G and beyond Systems». In: *Applied Sciences* 12.13 (2022). ISSN: 2076-3417. DOI: 10.3390/app12136617. URL: <https://www.mdpi.com/2076-3417/12/13/6617>.
- [5] Arunkumar Arulappan, Gunasekaran Raja, Kalpdrum Passi e Aniket Mahanti. «Optimization of 5G/6G Telecommunication Infrastructure through an NFV-Based Element Management System». In: *Symmetry* 14.5 (2022). ISSN: 2073-8994. DOI: 10.3390/sym14050978. URL: <https://www.mdpi.com/2073-8994/14/5/978>.
- [6] Abeer Ibrahim e Fazirulhisyam Hashim. «An architecture of 5G based on SDN NV wireless network». In: *Indonesian Journal of Electrical Engineering and Computer Science* 14 (mag. 2019), pp. 725–734. DOI: 10.11591/ijeeecs.v14.i2.pp725-734.
- [7] Gaurav Gangwal e Kevin Gray. *The 5G Core Network Demystified*. 2023. URL: <https://infohub.delltechnologies.com/it-it/p/the-5g-core-network-demystified/>.

- [8] CHANCE 5G. *La qualité de vie augmente avec chaque nouvelle génération de téléphonie mobile*. 2020. URL: <https://chance5g.ch/fr/articles/la-qualit%C3%A9-de-vie-augmente-avec-chaque-nouvelle-g%C3%A9n%C3%A9ration-d-e-t%C3%A9phonie-mobile/>.
- [9] ITU-R. *Framework and overall objectives of the future development of IMT for 2030 and beyond*. DRAFT NEW RECOMMENDATION. International Telecommunication Union (ITU-R), giu. 2023.
- [10] Vasileios Rekkas, Sotirios Sotirodis, Panagiotis Sarigiannidis, Shaohua Wan, George Karagiannidis e Sotirios Goudos. «Machine Learning in Beyond 5G/6G Networks—State-of-the-Art and Future Trends». In: *Electronics* 10 (nov. 2021), p. 2786. DOI: 10.3390/electronics10222786.
- [11] Amazon Web Services. *What is Digital Twin?* <https://aws.amazon.com/what-is/digital-twin/>. 2025.
- [12] TechSpot. *10 Best Fictional Technologies*. Feb. 2023. URL: <https://www.techspot.com/article/2213-10-best-fictional-technologies/>.
- [13] The Hollywood Reporter. *Berlin Works Taking Star Wars*. Gen. 2015. URL: <https://www.hollywoodreporter.com/movies/movie-news/berlin-works-taking-star-wars-768530/>.
- [14] Niral Networks. *6G Powered Robots: The Future of Robotics*. <https://www.linkedin.com/pulse/6g-powered-robots-future-robotics-niral-networks-ros/>. 2024.
- [15] Irian Pupo, Alejandro Santoyo-Gonzalez e Cristina Cervelló-Pastor. «A Framework for the Joint Placement of Edge Service Infrastructure and User Plane Functions for 5G». In: *Sensors* 19 (set. 2019). DOI: 10.3390/s19183975.
- [16] L. Ismail e R. Buyya. «Artificial intelligence applications and self-learning 6G networks for smart cities digital ecosystems: Taxonomy, challenges, and future directions». In: *Sensors* 22.15 (2022), p. 5750. DOI: 10.3390/s22155750.
- [17] Satyabrata Mishra. *Deep Reinforcement Learning: Intuitions*. https://www.linkedin.com/pulse/deep-reinforcement-learning-intuitions-satyabrata-mishra?trk=public_post. 2023.
- [18] Stefano V. Albrecht, Filippos Christianos e Lukas Schäfer. «Chapter 2: Reinforcement Learning». In: *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. Massachusetts Institute of Technology, 2022, pp. 24–28.

- [19] Muddsair Sharif, Gero Lückemeyer e Huseyin Seker. «Context Aware - Resource Optimality in Electric Vehicle Smart2Charge Application. A Deep Reinforcement Learning Base Approach». In: *IEEE Access* PP (gen. 2023), pp. 1–1. DOI: 10.1109/ACCESS.2023.3305966.
- [20] OpenAI. *Spinning Up in Deep RL*. https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html.
- [21] Richard S. Sutton, David McAllester, Satinder Singh e Yishay Mansour. «Policy gradient methods for reinforcement learning with function approximation». In: *Proceedings of the 13th International Conference on Neural Information Processing Systems*. NIPS'99. Denver, CO: MIT Press, 1999, pp. 1057–1063.
- [22] Samina Amin. *Deep Q-Learning (DQN)*. Set. 2024. URL: <https://medium.com/@samina.amin/deep-q-learning-dqn-71c109586bae>.
- [23] Utsav Desai. «Q-learning: A value-based reinforcement learning algorithm». In: *Medium* (2021). URL: <https://medium.com/intro-to-artificial-intelligence/q-learning-a-value-based-reinforcement-learning-algorithm-272706d835cf>.
- [24] SmartLab AI. «Reinforcement Learning algorithms — an intuitive overview». In: *Medium* (2021). URL: <https://smartlabai.medium.com/reinforcement-learning-algorithms-an-intuitive-overview-904e2dff5bbc>.
- [25] Lorenzo Canese, Gian Carlo Cardarilli, Luca Di Nunzio, Rocco Fazzolari, Daniele Giardino, Marco Re e Sergio Spanò. «Multi-Agent Reinforcement Learning: A Review of Challenges and Applications». In: *Applied Sciences* 11.11 (2021). ISSN: 2076-3417. DOI: 10.3390/app11114948. URL: <https://www.mdpi.com/2076-3417/11/11/4948>.
- [26] Luciano Miuccio, Salvatore Riolo, Sumudu Samarakoon, Daniela Panno e Mehdi Bennis. *Learning Generalized Wireless MAC Communication Protocols via Abstraction*. 2022. arXiv: 2206.06331 [cs.NI]. URL: <https://arxiv.org/abs/2206.06331>.
- [27] Tianxu Li, Kun Zhu, Nguyen Cong Luong, Dusit Niyato, Qihui Wu, Yang Zhang e Bing Chen. *Applications of Multi-Agent Reinforcement Learning in Future Internet: A Comprehensive Survey*. 2022. arXiv: 2110.13484 [cs.AI]. URL: <https://arxiv.org/abs/2110.13484>.

- [28] Annie Wong, Thomas Bäck, Anna Kononova e Aske Plaat. *Multiagent Deep Reinforcement Learning: Challenges and Directions Towards Human-Like Approaches*. Giu. 2021. DOI: 10.48550/arXiv.2106.15691.
- [29] Luciano Miuccio, Salvatore Riolo, Sumudu Samarakoon, Mehdi Bennis e Daniela Panno. «On Learning Generalized Wireless MAC Communication Protocols via a Feasible Multi-Agent Reinforcement Learning Framework». In: *IEEE Transactions on Machine Learning in Communications and Networking* 2 (2024), pp. 298–317. DOI: 10.1109/TMLCN.2024.3368367.
- [30] Hugging Face. *Deep Q-Learning Algorithm - Deep Reinforcement Learning Course*. URL: <https://huggingface.co/learn/deep-rl-course/en/unit3/deep-q-algorithm>.
- [31] Shruti Dhumne. *Deep Q-Network (DQN)*. 2023. URL: <https://medium.com/@shruti.dhumne/deep-q-network-dqn-90e1a8799871>.
- [32] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen e Yi Wu. *The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games*. 2022. arXiv: 2103.01955 [cs.LG]. URL: <https://arxiv.org/abs/2103.01955>.
- [33] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang e Joel S. Emer. *Efficient Processing of Deep Neural Networks*. Chapter 3. Morgan & Claypool Publishers, 2017.
- [34] Maarten Grootendorst. *A Visual Guide to Quantization*. 2024. URL: <https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization>.
- [35] Ravi Kishore Kodali, Yatendra Prasad Upreti e Lakshmi Boppana. «A Quantization Approach for the Reduced Size of Large Language Models». In: *2024 16th International Conference on Knowledge and Smart Technology (KST)*. 2024, pp. 144–148. DOI: 10.1109/KST61284.2024.10499664.
- [36] Florian Algo. *Model Quantization 2: Uniform and Non-Uniform Quantization*. 2024. URL: https://medium.com/@florian_algo/model-quantization-2-uniform-and-non-uniform-quantization-47ca5b5d3ec0.
- [37] Encord. *Mean Square Error (MSE) - Glossary*. URL: <https://encord.com/glossary/mean-square-error-mse>.
- [38] Datadeep. *Cose la model compression nei modelli di deep learning*. 2023. URL: <https://datadeep.it/2023/03/22/cose-la-model-compression-nei-modelli-di-deep-learning>.

- [39] IBM. *Quantization*. 2023. URL: <https://www.ibm.com/it-it/think/topics/quantization>.
- [40] PyTorch. *Quantization in Practice*. 2023. URL: <https://pytorch.org/blog/quantization-in-practice/>.
- [41] PyTorch. *torch.ao.quantization.quantizedynamic*. 2023. URL: https://pytorch.org/docs/stable/generated/torch.ao.quantization.quantize_dynamic.html.
- [42] Pierre-Emmanuel Novac, Ghouthi Boukli, Alain Pegatoquet, Benoit Miramond e Vincent Gripon. «Quantization and Deployment of Deep Neural Networks on Microcontrollers». In: *Sensors (Basel, Switzerland)* 21 (apr. 2021). DOI: 10.3390/s21092984.
- [43] Towards Data Science. *Inside Quantization-Aware Training*. 2023. URL: <https://towardsdatascience.com/inside-quantization-aware-training-4f91c8837ead>.
- [44] Brian Chmiel, Liad Ben-Uri, Moran Shkolnik, Elad Hoffer, Ron Banner e Daniel Soudry. *Neural gradients are near-lognormal: improved quantized and sparse training*. 2020. arXiv: 2006.08173 [cs.CV]. URL: <https://arxiv.org/abs/2006.08173>.
- [45] Enrico Russo, Elio Vinciguerra, Giuseppe Ascia e Daniela Panno. «DNN Hardware Accelerator Selection for Feasible Deployment of MARL-Based MAC Protocols in Industrial IoT Networks». In: set. 2024, pp. 443–448. DOI: 10.1109/RTSI61910.2024.10761172.
- [46] Wikipedia contributors. *Hyperbolic functions*. 2025. URL: https://en.wikipedia.org/wiki/Hyperbolic_functions.
- [47] Geek3. *Hyperbolic Tangent function plot*. 2008. URL: https://commons.wikimedia.org/wiki/File:Hyperbolic_Tangent.svg.
- [48] Tabark Alkhaldi, Noor Essam, Ahmed Al-khazaali, Marwa Alhamdany, Baqar Hataf, Ali Ramadhan e Ali TaeiZadeh. «A Survey Study of the Deep Learning for Convolutional Neural Network Architecture». In: *BIO Web of Conferences* 97 (apr. 2024), p. 00083. DOI: 10.1051/bioconf/20249700083.
- [49] PyTorch. *torch.ao.quantization.prepare*. 2023. URL: <https://pytorch.org/docs/stable/generated/torch.ao.quantization.prepare.html>.
- [50] PyTorch. *torch.ao.quantization.convert*. 2023. URL: <https://pytorch.org/docs/stable/generated/torch.ao.quantization.convert.html>.

- [51] YouMath. *Media, Moda e Mediana*. Accesso: 17 febbraio 2025. 2024. URL: <https://www.youmath.it/lezioni/algebra-elementare/lezioni-di-algebra-e-aritmetica-per-scuole-medie/2064-media-modà-mediana.html>.