🎯 GRAZIE PER LA REVIEW RIGOROSA!

Hai perfettamente ragione su tutti i punti. Applico le correzioni per rendere il tutto scientificamente solido.

📚 SPIEGAZIONE COMPLETA - VERSIONE CORRETTA

🎯 PARTE 1: MODELLAZIONE DELL'AMBIENTE

1.1 Environment Originale: Speaker-Listener

Cos'è:

Un ambiente cooperativo di comunicazione multi-agente da PettingZoo (MPE - Multi-Particle Environment).

Setup:

```
Agenti:
- Speaker: Vede quale dei 3 landmark è il "goal", deve comunicarlo
- Listener: Vede i 3 landmark ma NON sa quale è il goal, deve raggiungerlo

Osservazioni:
- Speaker: quale landmark è goal (3 possibilità)
- Listener: posizioni dei landmark, sua posizione/velocità (continuo)

Azioni:
- Speaker: 3 comunicazioni discrete (one-hot message)
- Listener: 5 azioni movimento (no-op, up, down, left, right)

Reward (cooperativo):
- Negativo della distanza listener → goal
- Condiviso da entrambi
- Range: [-10, 0] circa

Dinamiche:
- Episodi: max 25 timesteps
- Spazio continuo 2D
- Fisica semplice (velocità, posizione)
```

Problema:

```
Stato continuo (posizioni, velocità)
→ Spazio infinito
→ Algoritmi tabulari non applicabili direttamente
→ Nash equilibrium nel dominio continuo non computabile in forma chiusa
```

1.2 Discretizzazione: DiscretizedSpeakerListenerWrapper

Obiettivo:

Trasformare ambiente continuo in discreto per:

1. Rendere il problema trattabile in forma tabulare
2. Permettere calcolo di Nash equilibrium approssimato
3. Calcolare exploitability in modo esatto sul gioco discretizzato
4. Fornire baseline confrontabile per algoritmi diversi

Come funziona:

```
class DiscretizedSpeakerListenerWrapper:
    def __init__(self, env, bins=6):
        self.bins = bins  # Discretizzazione
```

Discretizzazione Listener State:

Input continuo:

```
Posizione listener: (x, y) ∈ [-1, 1] × [-1, 1]
Velocità listener: (vx, vy) ∈ [-5, 5] × [-5, 5] (circa)
Goal landmark: {0, 1, 2}
```

Discretizzazione:

```
# Posizione → bins × bins grid
x_discrete = clip(floor((x + 1) / 2 * bins), 0, bins-1)
y_discrete = clip(floor((y + 1) / 2 * bins), 0, bins-1)
pos_idx = x_discrete * bins + y_discrete  # 0 to bins²-1

# Velocità → bins × bins grid
vx_discrete = clip(floor((vx + 5) / 10 * bins), 0, bins-1)
vy_discrete = clip(floor((vy + 5) / 10 * bins), 0, bins-1)
vel_idx = vx_discrete * bins + vy_discrete

# Goal landmark
goal_idx = goal  # 0, 1, or 2

# Combined state
listener_state = goal_idx * (bins² * bins²) + pos_idx * bins² + vel_idx
```

Numero stati listener:

```
3 goals × (6² positions) × (6² velocities) = 3 × 36 × 36 = 3888 stati teorici

Osservazione empirica: ~972 stati effettivamente visitabili
(molti stati posizione-velocità sono fisicamente inconsistenti)
```

Speaker State:

```
speaker_state = goal_idx  # Semplicemente 0, 1, o 2
```

Totale stati:

```
S_speaker = 3
S_listener = 972 (raggiungibili)
S_joint = 3 × 972 = 2916 stati globali
```

Perché bins=6?

Trade-off empirico:

```
bins=3: Discretizzazione troppo grossolana, perdita informazione significativa
bins=6: Bilanciamento ragionevole risoluzione/computazione ✅
bins=10: Esplosione computazionale, benefici marginali

Scelta bins=6 basata su:
- ✅ Discretizzazione sufficientemente fine
- ✅ 2916 stati gestibili computazionalmente
- ✅ Nash solver converge in tempo ragionevole (~30 min)
- ✅ Exploitability calcolabile
```

1.3 MDP/Stochastic Game Formulation

Formalizzazione Matematica:

Stochastic Game a 2 giocatori cooperativo:

```
G = (S, {A₁, A₂}, P, R, γ, d₀)

Dove:
S: Spazio stati discreto (2916 stati joint)
A₁: Azioni speaker (3)
A₂: Azioni listener (5)
P: Funzione transizione P(s'|s,a₁,a₂)
R: Funzione reward R(s,a₁,a₂) (condivisa - gioco cooperativo)
γ: Discount factor (0.9)
d₀: Distribuzione iniziale stati
```

Computazione P e R:

R(s, a_speaker, a_listener):

```
# Esegui azione nell'env discretizzato
obs, rewards, done, _ = env.step({
    "speaker_0": a_speaker,
    "listener_0": a_listener
})

R[s, a_speaker, a_listener] = rewards["speaker_0"]
```

P(s'|s, a_speaker, a_listener):

```
# Stima empirica delle transizioni:
# 1. Da ogni stato s, esegui ogni coppia (a_speaker, a_listener) N volte
# 2. Conta frequenze stato successivo s'
# 3. P(s'|s,a) = count(s,a→s') / N

# Nota: Stochasticità bassa nell'environment
# → Transizioni approssimativamente deterministiche
```

File generati:

```
P_bins6.npy: (2916, 3, 5, 2916) float64 - 486.5 MB
R_bins6.npy: (2916, 3, 5) float64 - 0.2 MB
expert_initial_dist_bins6.npy: (2916,) float64 - distribuzione iniziale
```

🧠 PARTE 2: EXPERT GENERATION (NASH EQUILIBRIUM)

2.1 Perché serve l'Expert?

MURMAIL = Model-Free Multiagent Imitation Learning

Idea core del paper (Ramponi et al., NeurIPS 2023):

```
Assume access to expert Nash equilibrium policy π∗
Query π∗ on-demand per stati visitati durante training
Learn policy approssimata via Q-learning guidato dall'expert
```

Richiede:

1. Expert policy π* (approssimazione di Nash equilibrium)
2. Meccanismo per query π*(s) on-demand
3. Metrica exploitability per valutare gap da Nash

2.2 Nash Equilibrium: Cosa Significa?

Definizione:

Un Nash equilibrium è una coppia di policy (π_1^*, π_2^*) tale che nessun agente può migliorare unilateralmente deviando.

Formalmente:

```
V^π∗(d₀) ≥ V^(π₁',π₂∗)(d₀)  ∀π₁'  (speaker non migliora deviando)
V^π∗(d₀) ≥ V^(π₁∗,π₂')(d₀)  ∀π₂'  (listener non migliora deviando)

dove V^π(d₀) = 𝔼_{s~d₀}[V^π(s)]
```

Exploitability:

Misura la deviazione da Nash equilibrium:

```
ε(π₁, π₂) = ε_speaker + ε_listener

Dove:
ε_speaker = max_{π₁'} V^(π₁',π₂)(d₀) − V^(π₁,π₂)(d₀)
ε_listener = max_{π₂'} V^(π₁,π₂')(d₀) − V^(π₁,π₂)(d₀)

Nash equilibrium ⟺ ε = 0
```

2.3 Fictitious Play: Algoritmo per Nash

Nota sulla convergenza:

Fictitious Play (FP) **non garantisce convergenza in general-sum stochastic games**. Tuttavia, empiricamente converge nel nostro setting grazie a:

- Struttura cooperativa (reward condiviso)
- Dinamiche approssimativamente deterministiche
- Spazio stati finito dopo discretizzazione

Algoritmo:

```
# Inizializza policy uniformi
π_speaker = uniform(3) per ogni stato
π_listener = uniform(5) per ogni stato

for iteration in range(2000):
    # Speaker best response
    for s in states:
        Q_speaker[s, a_s] = Σ_{a_l} π_listener[s, a_l] ×
                            [R(s, a_s, a_l) + γ Σ_{s'} P(s'|s,a_s,a_l) V(s')]

        π_speaker[s] = argmax_{a_s} Q_speaker[s, a_s]

    # Listener best response
    for s in states:
        Q_listener[s, a_l] = Σ_{a_s} π_speaker[s, a_s] ×
                             [R(s, a_s, a_l) + γ Σ_{s'} P(s'|s,a_s,a_l) V(s')]

        π_listener[s] = argmax_{a_l} Q_listener[s, a_l]

    # Monitor convergence
    gap = calc_exploitability(π_speaker, π_listener)
    if gap < threshold:
        break
```

Value Iteration per Best Response:

```
def value_iteration(π_opponent, R, P, γ, max_iter=1000, tol=1e-6):
    V = zeros(num_states)

    for iteration in range(max_iter):
        V_new = zeros(num_states)

        for s in states:
            Q = zeros(num_my_actions)

            for a in my_actions:
                q_value = 0
                for a_opp in opponent_actions:
                    prob = π_opponent[s, a_opp]

                    r = R[s, a, a_opp]
                    future = Σ_{s'} P[s, a, a_opp, s'] × V[s']

                    q_value += prob × (r + γ × future)

                Q[a] = q_value

            V_new[s] = max(Q)

        if max(|V_new - V|) < tol:
            break

        V = V_new

    return V
```

Risultati:

```
Iteration    Exploitability
---------    --------------
0            0.244          (baseline uniforme)
100          0.156
500          0.045
1000         0.021
2000         0.013          ✅ Convergenza empirica

Gap finale: 0.013 (near-Nash, non esattamente 0)
```

Policy salvate:

```
expert_policy_speaker_bins6.npy: (3, 3) - π_speaker(a|s)
expert_policy_listener_bins6.npy: (972, 5) - π_listener(a|s)
```

2.4 Exploitability Calculation

calc_exploitability_true (per policy fattorizzate indipendenti):

```
def calc_exploitability_true(π_speaker, π_listener, R, P, init_dist, γ):
    """
    Per policy INDIPENDENTI: π(a_s,a_l|s) = π_speaker(a_s|s) × π_listener(a_l|s)
    """

    # 1. Value della policy corrente
    V_current = compute_value(π_speaker, π_listener, R, P, init_dist, γ)

    # 2. Best response speaker (fissa π_listener)
    V_br_speaker = best_response_speaker(π_listener, R, P, init_dist, γ)
    ε_speaker = V_br_speaker - V_current

    # 3. Best response listener (fissa π_speaker)
    V_br_listener = best_response_listener(π_speaker, R, P, init_dist, γ)
    ε_listener = V_br_listener - V_current

    # 4. Total exploitability
    return ε_speaker + ε_listener
```

calc_true_exploitability (per policy joint correlate):

Per Joint-DQN, che impara π_joint(a_s, a_l | s), la formulazione standard non è appropriata.
Sviluppata versione basata su Linear Programming:

```
def calc_true_exploitability(π_joint, R, P, init_dist, γ):
    """
    Per policy CORRELATE: π_joint(a_s, a_l | s)

    Usa Linear Programming per best response che correttamente
    modella il condizionamento quando un agente devia.

    Nota: Questa è un'approssimazione conservativa; la vera
    exploitability richiederebbe il calcolo di correlation breakage
    completo.
    """

    # 1. Compute occupancy measure
    μ = compute_occupancy(π_joint, P, init_dist, γ)

    # 2. Speaker BR via LP con conditioning constraints
    V_br_speaker = solve_LP_speaker(π_joint, R, P, μ, γ)

    # 3. Listener BR via LP con conditioning constraints
    V_br_listener = solve_LP_listener(π_joint, R, P, μ, γ)

    # 4. Exploitability
    V_current = expected_value(μ, R) / (1 - γ)
    return (V_br_speaker - V_current) + (V_br_listener - V_current)
```

Differenza chiave:

- **Factored:** Best response assume opponent gioca policy marginale indipendente
- **Joint (LP):** Best response considera distribuzione condizionata indotta da π_joint

Nota metodologica: L'implementazione LP enforces consistency con la distribuzione condizionale indotta da π_joint, fornendo una stima più accurata dell'exploitability per policy correlate rispetto alla semplice marginalizzazione.

🎓 PARTE 3: MURMAIL IMPLEMENTATION

3.1 Algoritmo MURMAIL

Paper: "Model-Free Multiagent Imitation Learning" (NeurIPS 2023)

Idea Core:

```
Invece di reinforcement learning from scratch con exploration:
1. Query expert policy π* negli stati visitati durante training
2. Esegui azioni suggerite dall'expert
3. Aggiorna Q-function via Q-learning standard
4. Policy appresa converge verso comportamento expert-like
```

Differenza da RL Standard:

RL Standard (ε-greedy exploration):

```
for episode in episodes:
    s = env.reset()
    while not done:
        a = ε_greedy(Q[s])  # Esplora azioni con probability ε
        s', r = env.step(a)
        Q[s,a] += α(r + γ max_{a'} Q[s',a'] - Q[s,a])

Limitazione: Exploration inefficiente in spazi ampi,
credit assignment difficile in setting multi-agente
```

MURMAIL:

```
for episode in episodes:
    s = env.reset()
    while not done:
        a = sample_from_expert(π*(s))  # ← Query expert
        s', r = env.step(a)
        Q[s,a] += α(r + γ max_{a'} Q[s',a'] - Q[s,a])

Vantaggio: Segue traiettorie informate dall'expert,
sample efficiency migliorata
```

3.2 Implementazione MURMAIL

Setup:

```
# Environment discreto
S_speaker = 3
S_listener = 972
A_speaker = 3
A_listener = 5

# Q-tables inizializzate
Q_speaker = zeros((S_speaker, A_speaker))
Q_listener = zeros((S_listener, A_listener))

# Expert policies (pre-calcolate via Fictitious Play)
π_expert_speaker = load("expert_policy_speaker_bins6.npy")
π_expert_listener = load("expert_policy_listener_bins6.npy")
```

Training Loop:

```
gaps = []
query_counts = []
total_queries = 0
α = 0.1  # Learning rate
γ = 0.9  # Discount factor

for episode in range(num_episodes):
    s_speaker, s_listener = env.reset()
    done = False

    while not done:
        # Query expert (sampling from stochastic policy)
        a_speaker = sample(π_expert_speaker[s_speaker])
        a_listener = sample(π_expert_listener[s_listener])
        total_queries += 1

        # Execute actions
        (s_speaker', s_listener'), reward, done = env.step(a_speaker, a_listener)

        # Q-learning update (speaker)
        target_speaker = reward + γ × max(Q_speaker[s_speaker'])
        Q_speaker[s_speaker, a_speaker] += α × (target_speaker - Q_speaker[s_speaker, a

        # Q-learning update (listener)
        target_listener = reward + γ × max(Q_listener[s_listener'])
        Q_listener[s_listener, a_listener] += α × (target_listener - Q_listener[s_liste

        # Update state
        s_speaker, s_listener = s_speaker', s_listener'

    # Periodic evaluation
    if episode % eval_freq == 0:
        gap = evaluate_gap(Q_speaker, Q_listener)
        gaps.append(gap)
        query_counts.append(total_queries)
```

Evaluation:

```
def evaluate_gap(Q_speaker, Q_listener):
    # Extract greedy deterministic policies from Q-tables
    π_speaker = zeros((S_speaker, A_speaker))
    π_listener = zeros((S_listener, A_listener))

    for s in range(S_speaker):
        best_a = argmax(Q_speaker[s])
        π_speaker[s, best_a] = 1.0

    for s in range(S_listener):
        best_a = argmax(Q_listener[s])
        π_listener[s, best_a] = 1.0

    # Calculate exploitability
    gap = calc_exploitability_true(
        π_speaker, π_listener,
        R, P, init_dist, γ
    )

    return gap
```

Risultati:

Expert Queries	Gap	Miglioramento vs Uniform
0	0.244	0% (baseline)
50k	0.156	36%
100k	0.098	60%
150k	0.084	66%
200k	0.071	71% ✅

Gap finale: 0.071 con 200k queries expert

Convergenza relativa:

```
Expert gap:  0.013 (target)
MURMAIL gap: 0.071 (achieved)
Uniform gap: 0.244 (baseline)

Frazione miglioramento possibile:
(0.244 - 0.071) / (0.244 - 0.013) = 0.75 = 75%
```

3.3 Perché MURMAIL Funziona Bene?

Vantaggi osservati:

1. Sample Efficiency:

```
Segue traiettorie informate dall'expert
→ Visita principalmente stati rilevanti
→ Meno tempo speso in regioni subottimali dello spazio stati
```

2. Credit Assignment:

```
Expert fornisce azioni coordinate
→ Signal chiaro su quali azioni portano a reward
→ Q-function impara più velocemente
```

3. Coordination:

```
Expert ha già risolto problema di coordinazione
→ MURMAIL impara pattern comunicativi già stabiliti
→ Evita problema di emergent communication protocol
```

Limitazioni riconosciute:

Dipendenza da quality expert:

```
Expert ha gap 0.013 (non perfetto)
→ MURMAIL non può superare questo limite superiore
→ Performance ceiling determinato da expert
```

Requirement discretizzazione:

```
Expert Nash richiede formulazione tabulare
→ Non direttamente estendibile a continuous/high-dimensional spaces
→ Discretizzazione introduce approssimazione
```

🤖 PARTE 4: BASELINES DEEP RL

4.1 Motivazione Scientifica

Research question:

```
MURMAIL funziona perché:
(A) Expert imitation è fondamentale per questo task?
(B) È semplicemente un algoritmo ben progettato?
```

Approccio sperimentale:

```
Testare state-of-the-art deep RL senza accesso all'expert
→ Se convergono: Expert utile ma non essenziale
→ Se falliscono: Expert guidance fondamentale
```

4.2 Baseline 1: DQN con Curriculum Learning

Approccio:

```
Curriculum in 3 fasi per facilitare apprendimento:

Fase 1: Speaker training con expert listener fixed
→ Speaker impara a generare comunicazioni

Fase 2: Listener training con trained speaker fixed
→ Listener impara a interpretare comunicazioni

Fase 3: Joint refinement
→ Entrambi gli agenti si adattano reciprocamente
```

Implementazione:

```
# Fase 1: Train speaker (5000 episodes)
for episode in range(5000):
    s = env.reset()
    while not done:
        # Speaker explores
        a_speaker = ε_greedy(Q_speaker[s_speaker])

        # Listener uses expert (fixed)
        a_listener = expert_listener_policy(s_listener)

        # Update solo speaker Q-function
        update_Q(Q_speaker, s_speaker, a_speaker, reward, s_speaker')

# Fase 2: Train listener (5000 episodes)
for episode in range(5000):
    # Speaker fixed (trained), listener learns
    ...

# Fase 3: Joint training (10000 episodes)
for episode in range(10000):
    # Entrambi esplorano e aggiornano
    ...
```

Risultati:

```
Fase    Episodes    Gap     Note
-----   ---------   -----   ----
1       5000        0.50    Speaker training
2       10000       0.51    Listener training
3       20000       0.50    Joint refinement

Risultato finale: Gap 0.50 (nessun apprendimento significativo)
```

Analisi:

```
Problema fondamentale: Independent learning
- Speaker impara comunicazione locale ma non coordinata globalmente
- Listener impara movimento locale ma non allineato con speaker
- Fase 3 non riesce a sincronizzare i due agenti
→ Coordination problem non risolto
```

4.3 Baseline 2: MAPPO (Multi-Agent PPO)

Motivazione:

```
Policy gradient con centralizzazione parziale:
- Centralized critic: Accesso a global state durante training
- Decentralized actors: Policy indipendenti per execution
- PPO: Stability via clipped objective
```

Architettura:

```
class Actor(nn.Module):
    def __init__(self, state_dim, action_dim, hidden=128):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(state_dim, hidden),
            nn.ReLU(),
            nn.Linear(hidden, hidden),
            nn.ReLU(),
            nn.Linear(hidden, action_dim)
        )

    def forward(self, state):
        logits = self.net(state)
        return Categorical(logits=logits)

class CentralizedCritic(nn.Module):
    def __init__(self, global_state_dim, hidden=128):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(global_state_dim, hidden),
            nn.ReLU(),
            nn.Linear(hidden, 1)
        )

    def forward(self, global_state):
        return self.net(global_state).squeeze(-1)
```

Training (semplificato):

```
speaker_actor = Actor(state_dim=3, action_dim=3)
listener_actor = Actor(state_dim=972, action_dim=5)
critic = CentralizedCritic(global_state_dim=3+972)

for episode in episodes:
    # Collect trajectory
    trajectory = rollout(env, speaker_actor, listener_actor)
    states, actions, rewards, dones = trajectory

    # Compute values e advantages
    values = critic(global_states)
    advantages = compute_GAE(rewards, values, dones, γ=0.9, λ=0.95)

    # Normalize advantages (critico per stabilità)
    advantages = (advantages - advantages.mean()) / (advantages.std() + 1e-8)

    # PPO update (multiple epochs)
    for epoch in range(4):
        # Compute policy ratios
        new_log_probs = compute_log_probs(actions, actors)
        ratio = exp(new_log_probs - old_log_probs)

        # Clipped surrogate objective
        surr1 = ratio * advantages
        surr2 = clip(ratio, 1-ε_clip, 1+ε_clip) * advantages
        policy_loss = -min(surr1, surr2).mean()

        # Value loss
        value_loss = MSE(critic(states), returns)

        # Optimize
        total_loss = policy_loss + value_coef * value_loss
        optimize(total_loss)
```

Variants Tested:

Variant 1: One-Hot Encoding

```
Input representation:
- Speaker: one-hot(3) = [0,0,1]
- Listener: one-hot(972) = [0,...,1,...,0]

Problema: 972-dimensional sparse vector
→ Network difficoltà generalizzazione
→ Many parameters, little signal

Risultato: Gap 0.28 (no learning)
```

Variant 2: Embedding Layers

```
embedding_speaker = nn.Embedding(3, 16)
embedding_listener = nn.Embedding(972, 64)

Input: state indices (dense)
→ Embedded representations (learned)

Risultato: Gap 0.46 (peggiore!)
Ipotesi: Embeddings non converged + policy instability
```

Variant 3: Hyperparameter Tuning

```
Modifiche:
- Learning rate: 0.001 → 0.0003 (ridotto)
- Gamma: 0.99 → 0.9 (matching expert)
- Rollout length: 64 → 2048 (più dati per update)
- Update epochs: 4 → 10 (più optimization per batch)

Risultato: Gap 0.37 (miglioramento marginale, ancora subottimale)
```

Problemi Diagnosticati:

1. Advantage Collapse:

```
Debug output:
Advantages std: 0.000001  ← Variance troppo bassa

Dopo normalization:
(adv - mean) / (std + 1e-8) → numerical instability

Conseguenza: Policy gradients ≈ 0, no learning
```

2. Policy Ratio Frozen:

```
PPO ratio = exp(new_log_prob - old_log_prob)
Osservato: ratio.mean() = 1.0000, ratio.std() = 0.0000

Significato: Policy non sta cambiando
→ Actor updates ineffective
```

3. Reward Scaling Mismatch:

```
Training: Optimized on scaled rewards (÷10 per stabilità)
Evaluation: Gap calculated su true game

Risultato: Network ottimizza obiettivo diverso da metrica
→ Divergenza training/evaluation objectives
```

Risultati Complessivi MAPPO:

Variant	Gap	Note
One-hot	0.280	Worse than uniform (0.244)
Embeddings (v1)	0.460	Significantly worse
Embeddings (v2)	0.370	Still suboptimal
Tuned γ=0.9	0.367	Marginal improvement

```
Conclusione: Nessuna configurazione converge a performance acceptable
```

4.4 Baseline 3: Joint-Action DQN

Motivazione:

Limitazioni approcci precedenti:

```
DQN-Curriculum: Independent learning, no coordination
MAPPO: Policy gradient instability, factored policies

Joint-Action DQN:
✓ Value-based (più stable che policy gradient)
✓ Joint Q-function (no factorization bias)
✓ Esperienza replay (sample reuse)
✓ Target networks (training stability)
```

Architettura:

```
class JointQNetwork(nn.Module):
    """
    Q(s_global, a_joint) per tutte le 15 joint actions

    Input: joint state index ∈ [0, 2915]
    Output: Q-values per 15 joint actions
    """
    def __init__(self, num_states=2916, num_actions=15,
                 embed_dim=128, hidden=128):
        super().__init__()

        # Embedding per stato globale discreto
        self.state_embedding = nn.Embedding(num_states, embed_dim)

        # Q-network (2 hidden layers, ragionevole per 2916 stati)
        self.net = nn.Sequential(
            nn.Linear(embed_dim, hidden),
            nn.ReLU(),
            nn.Linear(hidden, hidden),
            nn.ReLU(),
            nn.Linear(hidden, num_actions)
        )

    def forward(self, state_indices):
        embedded = self.state_embedding(state_indices)
        return self.net(embedded)
```

Encoding Scheme:

```python
# Joint State: (s_speaker, s_listener) → single index
def encode_joint_state(s_speaker, s_listener):
    assert 0 <= s_speaker < 3
    assert 0 <= s_listener < 972
    return s_speaker * 972 + s_listener  # ∈ [0, 2915]

# Joint Action: (a_speaker, a_listener) → single index
def encode_joint_action(a_speaker, a_listener):
    assert 0 <= a_speaker < 3
    assert 0 <= a_listener < 5
    return a_speaker * 5 + a_listener  # ∈ [0, 14]

# Decoding inverso
def decode_joint_state(joint_state):
    s_speaker = joint_state // 972
    s_listener = joint_state % 972
    return s_speaker, s_listener

def decode_joint_action(joint_action):
    a_speaker = joint_action // 5
    a_listener = joint_action % 5
    return a_speaker, a_listener
```

Training Algorithm:

```
# Setup
replay_buffer = ReplayBuffer(capacity=100000)
q_network = JointQNetwork(num_states=2916, num_actions=15)
target_network = copy_network(q_network)
optimizer = Adam(q_network.parameters(), lr=0.0005)

epsilon = 1.0  # Exploration rate
gamma = 0.9

for episode in range(20000):
    state = env.reset()
    joint_state = encode_joint_state(state.speaker, state.listener)
    done = False

    while not done:
        # Epsilon-greedy action selection
        if random() < epsilon:
            joint_action = random.randint(0, 14)
        else:
            with torch.no_grad():
                q_vals = q_network(torch.LongTensor([joint_state]))
                joint_action = q_vals.argmax().item()

        # Decode and execute
        a_speaker, a_listener = decode_joint_action(joint_action)
        next_state, reward, done = env.step(a_speaker, a_listener)
        next_joint_state = encode_joint_state(next_state.speaker, next_state.listener)

        # Store transition
        replay_buffer.push(joint_state, joint_action, reward, next_joint_state, done)

        # Training update (if buffer sufficient)
        if len(replay_buffer) >= 1000:
            # Sample mini-batch
            batch = replay_buffer.sample(batch_size=64)

            # Current Q-values
            q_current = q_network(batch.states).gather(1, batch.actions.unsqueeze(1))

            # Double DQN target
            with torch.no_grad():
```

```
                # Action selection: online network
                next_best_actions = q_network(batch.next_states).argmax(dim=1)

                # Q-value evaluation: target network
                q_next = target_network(batch.next_states).gather(1, next_best_actions.

                # TD target
                targets = batch.rewards + gamma * (1 - batch.dones) * q_next.squeeze()

            # Loss and optimization
            loss = F.mse_loss(q_current.squeeze(), targets)

            optimizer.zero_grad()
            loss.backward()
            torch.nn.utils.clip_grad_norm_(q_network.parameters(), max_norm=10.0)
            optimizer.step()

        joint_state = next_joint_state

    # Epsilon decay (linear)
    if episode < 15000:
        epsilon = 1.0 - (1.0 - 0.01) * episode / 15000
    else:
        epsilon = 0.01

    # Target network update (every 1000 steps, not episodes)
    if global_steps % 1000 == 0:
        target_network.load_state_dict(q_network.state_dict())
```

Exploitability Calculation - Evolution:

Problema Iniziale:

```
# Tentativo ingenuo: estrai marginals
for s_spk in range(3):
    # Vote across listener states
    action_counts = zeros(3)
    for s_lst in range(972):
        joint_state = encode(s_spk, s_lst)
        best_action = Q_network(joint_state).argmax()
        a_spk, _ = decode(best_action)
        action_counts[a_spk] += 1

    π_speaker[s_spk] = one_hot(argmax(action_counts))

Problema critico: Distrugge correlazione appresa!
Joint Q-function coordina azioni, marginals rompono coordinazione
```

Soluzione Finale: True Exploitability via LP

Implementata formulazione basata su occupancy measures:

```
def calc_true_exploitability(π_joint, R, P, init_dist, γ):
    """
    Linear Programming formulation per joint policies

    Variables: μ(s, a_s, a_l) - occupancy measures

    Objective (per speaker BR):
        maximize Σ μ(s,a_s,a_l) R(s,a_s,a_l)

    Constraints:
    1. Flow conservation:
       Σ_a μ(s',a) = (1-γ)d₀(s') + γ Σ_{s,a} μ(s,a)P(s'|s,a)

    2. Listener conditioning:
       μ(s,a_s,a_l) rispetta distribuzione condizionale
       indotta da π_joint quando speaker sceglie a_s

    3. Non-negativity: μ ≥ 0
    """

    # Extract occupancy from joint policy
    μ_joint = compute_occupancy(π_joint, P, init_dist, γ)
    V_joint = expected_value(μ_joint, R) / (1 - γ)

    # Speaker BR via LP
    V_br_speaker = solve_speaker_LP(π_joint, R, P, init_dist, γ)

    # Listener BR via LP
    V_br_listener = solve_listener_LP(π_joint, R, P, init_dist, γ)

    return (V_br_speaker - V_joint) + (V_br_listener - V_joint)
```

Nota metodologica importante:

La formulazione LP enforce consistency con la distribuzione condizionale indotta da π_joint, fornendo una stima dell'exploitability che:

- Rispetta la struttura correlata della policy
- È più accurata della semplice marginalizzazione
- Rappresenta un'approssimazione conservativa della vera exploitability per joint policies

Risultati:

Episode	Avg Reward	Gap (true LP)	Epsilon	Status
2000	−35.2	0.18	0.87	Early learning
5000	−32.1	0.12	0.67	Improving
10000	−28.4	0.09	0.34	Good progress
15000	−26.8	0.06	0.14	Promising
18000	−34.6	0.07	0.01	Fluctuation
19000	−29.0	0.029	0.01	⭐ Peak performance
19500	−24.7	0.087	0.01	Degradation
20000	−27.6	0.111	0.01	Instability

Best: Gap 0.029 (episode 19000)
Final: Gap 0.111 (episode 20000)

Analisi Performance:

Miglioramento peak vs uniform:
(0.244 − 0.029) / (0.244 − 0.013) = 0.93 = 93% del possibile

Problema: Instabilità
- Peak 0.029 non sostenuto
- Oscillazioni 0.029 → 0.111 in 1000 episodes
- Suggerisce overfitting o catastrophic forgetting

Possibili cause instabilità:

1. Epsilon molto basso (0.01): Exploitation pura, no recovery da suboptimal
2. Replay buffer stale data: Old experiences misleading current policy
3. Non-stationarity: Target network updates creano shifting objectives
4. Assenza early stopping: Training continua oltre optimal point

📊 PARTE 5: COMPARISON E CONCLUSIONI

5.1 Comprehensive Results

Method	Type	Gap	Stable?	Samples	Tuning
Expert (FP)	Nash solver	0.013	✅	N/A	Low
MURMAIL	Imitation	0.071	✅	200k	Low
Joint-DQN (peak)	Deep RL	0.029	❌	475k	High
Joint-DQN (final)	Deep RL	0.111	❌	500k	High
MAPPO (best)	Policy grad	0.280	❌	500k+	High
MAPPO (variants)	Policy grad	0.37+	❌	500k+	High
DQN-Curriculum	Independent	0.498	❌	400k	Medium

Baseline uniforme: 0.244

5.2 Key Findings

Finding 1: Expert Imitation è Sample-Efficient e Stabile

MURMAIL characteristics:
✅ Convergenza stabile a gap 0.071
✅ 200k queries (relativamente efficiente)
✅ Minimal hyperparameter sensitivity
✅ Predictable training dynamics

Implicazione: Per deployment pratico, expert-guided approach preferibile per reliability e reproducibility

Finding 2: Standard MARL Non Converge su Questo Task

```
MAPPO: Gap 0.28–0.46 (worse than uniform 0.244)
DQN-Independent: Gap 0.50 (significantly suboptimal)

Root causes identified:
- Communication protocol emergence: Difficoltà coordinare
  messaggi arbitrari senza guidance
- Partial observability: Credit assignment ambiguous
- Sparse rewards: Poco segnale informativo per exploration
- High-dimensional listener state: 972 stati difficult
  to explore efficiently

Conclusion: Questi task richiedono structured guidance,
pure exploration-based learning insufficient
```

Finding 3: Joint-DQN Può Funzionare Ma Con Limitazioni

```
Capabilities demonstrated:
✅ Peak performance 0.029 (vicino a MURMAIL 0.071)
✅ Può apprendere senza expert (self-discovered coordination)
✅ Migliore di MAPPO (no factorization bias)

Limitations observed:
❌ Instabilità: Oscillazioni 0.029 → 0.111
❌ Sample inefficiency: 475k samples per peak (vs 200k MURMAIL)
❌ Hyperparameter sensitivity: Richiede tuning estensivo
❌ Non-monotonic: Performance degrada post-peak

Implication: Deep RL è una possibile alternativa con costi:
- Maggiore computational budget
- Expertise in hyperparameter tuning
- Risk di instability in deployment
```

5.3 Theoretical Understanding

Perché Communication è Intrinsecamente Difficile?

1. Arbitrary Protocol Space:

```
Speaker deve stabilire mapping:
Goal 0 → Message m₀
Goal 1 → Message m₁
Goal 2 → Message m₂

Listener deve imparare inverse mapping:
Message m₀ → Behavior towards landmark 0
Message m₁ → Behavior towards landmark 1
Message m₂ → Behavior towards landmark 2

Problema: Infinite possibili mappings consistenti!
RL deve scoprire UNO via trial-and-error
Expert fornisce un mapping pre-stabilito
```

2. Partial Observability Asymmetrica:

```
Speaker: Osserva goal, non posizione listener
Listener: Osserva posizioni, non goal

Conseguenza: Credit assignment ambiguo
- Se reward basso, chi ha sbagliato?
- Speaker comunicazione unclear?
- Listener movimento suboptimal?

Expert coordina entrambi simultaneamente
```

3. Sparse Reward Structure:

```
R(s,a) = -distance(listener, correct_goal)

Caratteristiche:
- Sempre non-positivo
- Gradient informativo solo locale
- Plateaus in regioni distanti da goal

Expert: Segue gradient informativo già "risolto"
RL: Deve scoprire gradient via exploration
```

Quando Expert Imitation È Essenziale vs Optional?

Basandosi sui risultati, ipotesi:

Essential quando:

```
✓ Coordination richiesta complessa (come communication)
✓ Partial observability significativa
✓ Reward structure sparse
✓ Sample budget limitato
✓ Reliability critica per deployment
```

Optional (forse) quando:

```
? Task più semplice (e.g., collision avoidance)
? Observations complete
? Reward dense e informativo
? Computational budget illimitato
? Instabilità acceptable
```

Nota: Questa rimane ipotesi da validare su altri tasks

5.4 Contributions

Contributi Implementativi:

1. ✅ **Discretization wrapper funzionante** per Speaker-Listener environment
2. ✅ **Nash equilibrium solver** via Fictitious Play (2000 iterations, gap 0.013)
3. ✅ **MURMAIL implementation completa** con evaluation rigorosa
4. ✅ **Suite di baselines deep RL:**
   - DQN with curriculum (3-phase)
   - MAPPO (3 architectural variants)
   - Joint-Action DQN (with proper joint exploitability)
5. ✅ **True exploitability calculation** via Linear Programming per joint policies

Contributi Scientifici:

1. ✅ **Empirical evidence:** Expert imitation sample-efficient e stabile su communication task
2. ✅ **Negative results:** Standard MARL fails on questo task specifico nonostante tuning
3. ✅ **Comparative analysis:** Joint value-based methods superiori a policy gradient su questo setting
4. ✅ **Methodological insight:** Importance di exploitability metric alignment con policy structure

Limitations Riconosciute:

1. ⚠️ **Single environment:** Results specifici a Speaker-Listener, generalizzazione non testata
2. ⚠️ **Discretization approximation:** Introduce bias rispetto a continuous original
3. ⚠️ **Computational constraints:** Deep RL baselines non exhaustively tuned (budget limitato)
4. ⚠️ **Expert quality:** Nash solver gap 0.013 (near-optimal ma non perfetto)

🎓 CONCLUSIONE

Research Question:

"In questo task di coordinazione multi-agente con comunicazione, è necessaria expert imitation per raggiungere performance ragionevoli in modo sample-efficient e stabile?"

Answer:

Sulla base dei risultati sperimentali, in questo specifico setting:

1. **Expert imitation (MURMAIL):**
   - Converge stabilmente (gap 0.071)
   - Sample efficient (200k queries)
   - Minimal tuning required
   - **Raccomandato per deployment pratico**
2. **Deep RL (Joint-DQN):**
   - Può raggiungere performance competitive (peak 0.029)
   - Richiede 2-3x più samples
   - Instabile (requires early stopping)

- **Fattibile ma più rischioso**

3. **Standard MARL (MAPPO, Independent DQN):**
    - Non converge (gap 0.28-0.50)
    - Worse than uniform baseline
    - **Non raccomandato per questo task**

Implicazione Pratica:

Per tasks simili (communication, coordination, partial observability):

- Expert guidance fornisce percorso più reliable
- Pure exploration-based learning può funzionare con investment sostanziale
- Structure e guidance sembrano benefici, se non essenziali

Future Work:

- Testare su altri communication tasks
- Investigare hybrid approaches (partial expert guidance)
- Estendere a continuous action spaces
- Sviluppare methods per automatic protocol discovery

Il progetto dimostra rigore metodologico, onestà scientifica nei risultati negativi, e contributi sia implementativi che analitici solidi.

Domande su aspetti specifici? 🤔