基于强化学习的多智能体路径规划

💻 **小组成员：**2252750 赵卓冰、2251941 王成威、2250758 林继申

1. 选题意义与价值
2. 模块设计
3. 算法设计
4. 实验过程
5. 实验结果与分析
6. 总结

1. 选题意义与价值

1.1 路径规划的现实痛点

1.1.1 A* 算法限制

- **多智能体死锁问题**：A* 算法在处理多智能体路径规划时，容易因路径交叉导致死锁，尤其是在高密度场景下，智能体之间的相互干扰会严重影响任务的完成效率。
- **动态环境适应性差**：A* 算法在面对动态障碍或实时拥堵时，难以快速调整路径，导致路径规划的实时性和灵活性不足。

1.1.2 DQN 的问题

- **训练收敛慢**：深度强化学习算法在多智能体路径规划中，训练过程往往收敛缓慢，难以在有限的时间内学习到最优策略。
- **奖励机制不足**：现有的奖励机制在多智能体场景中效果有限，导致策略学习效率低下，智能体难以快速适应环境变化。
- **Q 值过估计问题**：DQN 算法中常见的 Q 值过估计问题会影响决策的可靠性，导致智能体在复杂环境中做出次优决策。

1.2 项目目标与创新

本项目旨在通过深度强化学习技术，实现多智能体在网格化环境中的高效路径规划与拥堵控制。具体目标如下：

1.2.1 多智能体路径规划与拥堵控制

设计一个网格化的交通路由环境，在其中放置若干智能体（Agent）和各自的目标位置。智能体需要学会在遵守网格边界、避免与其他智能体同格或交叉移动（碰撞）的前提下，尽快从自己的起点到达目的地。同时，环境支持静态和动态障碍物的检测，智能体需要在规划路径时避开这些障碍物，减少拥堵现象。

1.2.2 基于深度强化学习的决策

使用 Double DQN 算法，让每个智能体独立地通过与环境不断交互，学习一套策略，动态调整路径，避免碰撞和拥堵。同时，使用多种常见路径规划方法，作为对照，展现引入架构的优越性。

1.2.3 奖励塑形以加速收敛

引入了奖励塑形机制，利用了曼哈顿调整奖励信号，引导智能体在探索过程中更倾向于选择那些能够使其更快接近目标的行动。让智能体朝着目标前进，减少不必要的迂回和探索，显著加快其学习收敛的速度，从而更高效地实现从起点到终点的路径规划。

1.2.4 可视化训练后策略

在训练结束后，用 ε = 0（完全贪心）策略，让所有智能体再跑一次，收集它们的移动轨迹并在一个网格图里绘制出来。通过这种方式，可以直观地看到各个智能体在学会避免碰撞后，是如何规划路径、到达各自目标的。验证算法的有效性，并为后续的改进提供直观的参考。

2. 模块设计

2.1 环境模块

TrafficRoutingEnv 类是本项目的核心环境模块，负责构建多智能体路径规划的仿真环境。该模块实现了一个基于离散网格的交通路由系统，支持多个智能体在同一环境中进行路径规划与避障操作。

2.1.1 核心功能

- **环境定义与管理**：本环境模块定义了一个 H×W 规格的离散网格环境，能够同时记录 N 个智能体的当前位置信息以及各自预设的目标位置。环境通过网格化的方式简化了连续空间中的复杂路径规划问题，为多智能体协同提供了标准化的操作平台。
- **动作处理与碰撞检测：**环境模块接收字典形式的动作输入，对每个智能体的移动意图进行统一处理。系统首先计算所有智能体的“期望新位置”，随后执行严格的碰撞检测机制：
  - **同格碰撞检测**：识别多个智能体试图移动到同一网格位置的冲突情况。

- **交换格子碰撞检测**：检测两个智能体相互交换位置的特殊冲突场景。当检测到碰撞时，系统将冲突方强制退回原始位置，确保环境状态的一致性和安全性。

- **奖励机制设计**：环境采用分层奖励机制来指导智能体的学习过程：
  - **基础奖励规则**：
    - 智能体首次到达目标位置：+10 分
    - 已到达目标后继续停留：0 分
    - 未到达目标的每个时间步：-1 分
  - **奖励塑形机制**：系统引入基于曼哈顿距离的潜力塑形（Potential Shaping），通过以下公式计算最终奖励：
    - $reward=reward\_raw+(\gamma \cdot \phi\_new-\phi\_old)$
    - 其中：
      - ϕ = -曼哈顿距离
      - γ 为折扣因子
      - ϕ_new 和 ϕ_old 分别表示动作执行后和执行前的潜力值

  这种设计有效引导智能体向目标方向移动，可以显著提升训练收敛速度。

- **状态冻结机制**：为避免已完成任务的智能体继续消耗计算资源或干扰其他智能体，系统实现了状态冻结功能。一旦某个智能体到达目标位置，其后续所有动作都被自动转换为“停留”指令，既不会产生位置变化，也不会获得负奖励。

2.1.2 主要方法说明

- **构造方法：init(self, grid_size, num_agents)**
  - **功能描述**：
    - 初始化环境参数并生成智能体的起始配置信息。
  - **参数说明**：
    - grid_size：网格环境的尺寸规格
    - num_agents：参与路径规划的智能体数量
  - **核心操作**：
    - 随机生成各智能体的起始位置（self._initial_positions）
    - 随机分配各智能体的目标位置（self.destinations）
    - 初始化环境内部状态变量
      - 当前位置记录：self.agent_positions
      - 到达状态字典

        - 时间步计数器
    - **设计特点**：起始位置和目标位置仅在环境构造时生成一次，确保训练过程中的一致性和可重现性。
- **重置方法：reset(self)**
    - **功能描述**：
        - 在每个训练回合开始时重置环境状态，恢复所有智能体到预设的初始配置。
    - **核心操作**：
        - 将所有智能体位置重置为初始位置
        - 清空所有智能体的到达状态标记
        - 重置时间步计数器为 0
        - 返回标准化的观测字典：{i:'position':...,'destination':...}
    - **关键特性**：该方法不会重新随机生成位置坐标，而是严格恢复到构造时确定的固定初始点，保证训练的稳定性和可比较性。
- **步进方法：step(self, actions)**
    - **功能描述**：
        - 执行一个完整的环境交互步骤，处理所有智能体的动作并返回环境反馈。
    - **输入参数**：
        - actions：字典格式的动作集合
        - 动作编码：0=向上，1=向下，2=向左，3=向右，4=停留
    - **执行流程**：
        - 状态记录阶段
            - 保存所有智能体的当前位置信息
        - 意图解析阶段
            - 根据动作指令计算各智能体的期望新位置
            - 对已到达目标的智能体强制保持静止状态
        - 冲突检测阶段
            - 执行交换碰撞检测：识别相互交换位置的智能体对
            - 执行同格碰撞检测：识别目标位置重叠的智能体组
            - 将所有冲突智能体回退至原始位置
        - 状态更新阶段
            - 更新智能体位置信息至 self.agent_positions

        - 递增时间步计数器
    - 奖励计算阶段
        - 计算各智能体的基础奖励值
        - 应用潜力塑形机制得到最终奖励 shaped_rewards[i]
        - 更新智能体到达状态标记
    - 终止判断阶段
        - 检查全局终止条件：所有智能体均已到达目标或时间步达到上限（50 步）
        - 设置相应的终止标志
- **返回值**：（next_obs， shaped_rewards， done， info={...}）
    - next_obs：更新后的环境观测信息
    - shaped_rewards：包含塑形奖励的奖励字典
    - done：全局终止标志
    - info：附加信息字典

2.2 强化学习架构

MADQNTrainer 类是本项目的核心训练模块，专门负责多智能体深度 Q 网络的训练管理工作。该模块的核心思想是为每个智能体维护独立的学习组件，通过去中心化的方式实现大规模多智能体系统的并行训练。这种架构避免了传统中心化训练中的通信瓶颈和计算复杂度问题，使得系统能够有效扩展到包含大量智能体的复杂环境中。

2.2.1 核心功能

- **分布式训练架构管理**：训练模块采用去中心化的架构设计，为 N 个智能体分别维护独立的训练组件集合。每个智能体配备专属的神经网络、目标网络、优化器以及经验回放缓冲区，确保各智能体的学习过程相互独立，避免了中心化训练可能带来的瓶颈问题。
- **ε-贪心策略实现**：系统采用经典的 ε-贪心探索策略来平衡探索与利用的关系。在训练过程中，智能体以 ε 的概率执行随机探索动作，以 1-ε 的概率选择当前策略认为最优的动作。该策略有效防止了训练过程中的过早收敛问题，保证了解决方案的多样性。
- **经验回放机制**：模块实现了基于经验回放的 Q-Learning 更新机制。系统持续收集五元组形式的训练样本（s, a, r, s', done_before），存储于各智能体的专属缓冲区中。通过随机抽样的方式打破样本间的时序相关性，显著提升了训练的稳定性和收敛效率。
- **目标网络同步策略**：为了缓解 Q-Learning 中的不稳定性问题，系统引入了目标网络机制。训练过程中定期将策略网络的参数同步至目标网络，在计算目标 Q 值时使用相对稳定的目标网络，有效减少了训练过程中的震荡现象。

2.2.2 主要组件详述

- **初始化方法：init(self, env, num_agents, state_dim, action_dim, buffer_capacity, batch_size, gamma, lr, target_update)**
  - **函数职责：**
    - 完成训练器的初始化配置，为每个智能体建立完整的训练基础设施。
  - **参数说明：**

参数名	类型	说明
env	TrafficRoutingEnv	交通路由环境实例
num_agents	int	参与训练的智能体数量
state_dim	int	状态向量维度（14 维：4 维坐标 + 9 维邻居信息 + 1 维距离）
action_dim	int	动作空间维度（固定为 5）
buffer_capacity	int	经验回放缓冲区容量
batch_size	int	批量训练样本大小
gamma	float	奖励折扣因子
lr	float	学习率参数
target_update	int	目标网络更新频率

  - **核心初始化操作**：对于每个智能体 i，系统执行以下初始化步骤：
    - 策略网络创建：policy_nets[i] = DQN(state_dim， action_dim)
    - 目标网络创建：target_nets[i] = DQN(state_dim， action_dim)
    - 网络权重同步：target_nets[i]。load_state_dict(policy_nets[i].state_dict())
    - 优化器配置：optimizers[i] = Adam(policy_nets[i]。parameters(), lr)
    - 缓冲区分配：replay_buffers[i] = ReplayBuffer(buffer_capacity)
- **动作选择方法：select_action(self, agent_id, state, eps, done)**
  - **函数职责：**
    - 根据当前状态和探索参数为指定智能体选择最优动作。
  - **执行逻辑：**
    - 终止状态处理：若智能体已到达目标位置（done=True），直接返回停留动作编码 4
    - 随机探索分支：以 ε 概率在动作空间中随机选择动作

        - 贪心利用分支：以 1-ε 概率通过策略网络计算 Q 值，选择 Q 值最大的动作
    - **技术特点**：
        - 自适应终止处理，避免无效计算
        - 平衡探索与利用，保证训练效果
        - 高效的神经网络推理
- **智能体优化方法：optimize_agent(self, agent_id)**
    - **函数职责**：
        - 对指定智能体执行一次 Q-Learning 参数更新。
    - **优化流程**：
        - 样本抽取阶段
            - 从指定智能体的经验缓冲区中随机抽取 batch_size 个训练样本
            - 样本格式：(s， a， r， s'， done)
        - Q 值计算阶段
            - 当前 Q 值：current_q = policy_net(s).gather(a)
            - 目标 Q 值：next_q = target_net(s').max(a')
            - 目标值计算：target_q = r + γ × next_q × (1 - done)
        - 损失计算与反向传播
            - 计算均方误差损失：MSE(current_q, target_q)
            - 执行反向传播更新策略网络参数
    - **算法优势**：
        - 基于时序差分的稳定学习
        - 批量处理提升训练效率
        - 目标网络减少训练震荡
- **目标网络更新方法：update_targets(self)**
    - **函数职责**：定期同步所有智能体的目标网络参数。
- **更新机制**：当训练回合数达到 target_update 的整数倍时，系统自动执行参数同步操作：target_nets[i].load_state_dict(policy_nets[i].state_dict())
- **设计原理**：目标网络的引入有效解决了 Q-Learning 中的“移动目标”问题，通过使用相对固定的目标网络计算目标 Q 值，显著提升了训练的稳定性。
- **主训练方法：train(self, num_episodes, max_steps, eps_start, eps_end, eps_decay)**
    - **函数职责**：

        - 执行完整的多智能体强化学习训练流程。
    - **参数配置**：
        - num_episodes：总训练回合数
        - max_steps：单回合最大步数
        - eps_start：初始探索率
        - eps_end：最终探索率
        - eps_decay：探索率衰减系数
    - **训练流程详述**：
        - 回合初始化阶段
            - obs = self.env.reset()
            - state_dict = obs_to_state(obs, self.env.grid_size)
            - done_dict = {i: False for i in range(self.num_agents)}
            - total_reward = 0
        - 交互学习循环：在每个时间步内执行以下操作序列：
            - 动作决策阶段
                - 为所有智能体并行执行 ε-贪心动作选择
                - 生成完整的动作字典用于环境交互
            - 环境交互阶段
                - 执行环境步进操作获取反馈信息
                - 解析环境返回的状态、奖励和终止信号
            - 经验存储阶段
                - 将交互产生的五元组数据存入各智能体的回放缓冲区
                - 更新智能体到达状态标记
                - 累计回合总奖励
            - 网络优化阶段
                - 对所有智能体并行执行 Q-Learning 参数更新
                - 基于经验回放机制进行批量学习
            - 终止条件检查
                - 检查全局终止条件决定是否结束当前回合
        - 回合后处理阶段：
            - 奖励记录：将回合累积奖励添加至训练历史

- 探索率衰减：按照指数衰减策略更新 ε 值
- 目标网络同步：根据更新频率决定是否同步目标网络
- 训练监控：定期输出训练进度和性能指标

2.3 可视化模块

可视化模块负责在训练结束后，通过设置 ε = 0（完全贪心策略）执行一次完整的环境回合，展示所有智能体的路径规划结果。该模块收集每个智能体的移动轨迹，并将其绘制成实时动画，以网格地图的形式动态展示多智能体在环境中的协同避障与目标到达过程。

2.3.1 核心功能流程

- **环境初始化**
  - 初始化网格环境尺寸（H×W）。
  - 为每个智能体创建位置轨迹记录器（agent_trajectories）。
  - 初始化 done 状态字典，标记每个智能体是否已到达目标。
- **收集轨迹数据**
  - 环境重置至初始状态。
  - 利用贪心策略（ε = 0）迭代执行智能体动作。
  - 每一步计算智能体动作并更新环境状态。
  - 记录所有智能体在每一步的位置，直到所有智能体到达目标或达到最大步数（50 步）。
- **图形绘制与动态更新**
  - 绘制固定网格线作为背景。
  - 绘制障碍物（黑色填充方块）。
  - 使用方块（■）标记起点，使用星形（★）标记终点。
  - 每个智能体分配独立颜色，轨迹用折线动态呈现。
  - 添加步骤计数文本，实时更新显示当前步数
- **动画生成与保存**
  - 定义更新函数，根据当前帧更新所有智能体的位置与轨迹。
  - 利用 Matplotlib 的 FuncAnimation 生成动态图。
  - 支持实时展示与 GIF 导出。

2.3.2 核心作用

- **直观演示**：可视化模块的设计核心在于通过直观的动态展示，验证多智能体路径规划系统的有效性和可靠性。强化学习本身是一个“黑盒”过程，仅依赖奖励曲线或损失函数难以全面评估模型性能。动画能够直观展示多个智能体如何在复杂环境中有效避障、避免碰撞并顺利到达目标。
- **辅助开发**：它能够辅助开发者快速识别训练过程中的潜在问题，例如路径震荡、拥堵无法化解或智能体死锁。相比传统的数值指标，动态可视化更适用于性能对比、实验结果展示以及工业用户的信任建立。此外，在实际部署中，可视化模块不仅服务于模型验证阶段，也可以作为系统运行时的监控工具，为调度优化提供实时反馈。
- **性能评估**：该模块的主要作用是直观展示智能体路径规划效果，辅助评估算法的避障能力、路径效率和收敛质量。它能够实时呈现多智能体在网格环境中的运动轨迹，反映训练后的策略质量。作为系统监控的一部分，提供了任务执行的动态回放，增强系统的可操作性与可信度。

3. 算法设计

3.1 概览

模块	功能	算法核心
Astar	静态 A* 搜索，无障碍，作为基准	启发式搜索
DNQ	无障碍多智能体 DQN 训练	DQN + Potential Shaping
DNQObstacle	加入障碍物约束的 DQN	DQN + Obstacle-Aware
DoubleDNQ	加入障碍物 + Dueling + Double DQN，训练更稳定	Dueling DQN + Double DQN

3.2 对照算法设计

3.2.1 Astar.py —— 基线路径规划算法

- **核心功能**：
  - 实现标准 A* 搜索算法，计算无障碍情况下的最短路径。
  - 模拟多智能体在静态路径上的移动，并检测简单碰撞（同格冲突）。
- **主要类**：AstarBaseline
  - astar_search()：单智能体路径规划。
  - run_baseline()：多智能体路径执行，带有碰撞检测。
- **架构特点**：
  - 不考虑障碍物，只在空网格上计算静态最短路径。
  - 无学习能力，属于规则基准。

3.2.2 DNQ.py —— 无障碍深度强化学习

- **核心功能**：
  - 基于 DQN 的多智能体路径规划。
  - 采用去中心化架构，每个智能体有独立的神经网络、目标网络和经验回放。
- **主要类**：
  - **TrafficRoutingEnv**
    - 构建无障碍的网格化交通环境。
    - 包含完整的动作解析、碰撞检测和奖励塑形（曼哈顿距离）。
  - **DQN**
    - 标准三层全连接神经网络。
  - **ReplayBuffer**
    - 经验回放缓冲区。
  - **MADQNTrainer**
    - 多智能体训练管理器，支持目标网络更新、ε 贪心策略、并行训练。
- **架构特点**：
  - 状态空间：14 维（4 维位置，9 维邻居占用，1 维归一化距离）。
  - 动作空间：5（上、下、左、右、停留）。
  - 无障碍物，纯基于智能体间的碰撞控制。

3.2.3 DNQObstacle.py —— 加入障碍物的 DQN 架构

- **核心功能**：
  - 在 DNQ 基础上，添加障碍物的约束。
  - 智能体需同时规避障碍物和其他智能体。
- **主要变化**：
  - TrafficRoutingEnv 新增障碍物列表（Obstacles）。
  - 状态空间保持 14 维，但邻居占用信息包含障碍物状态。
- **架构一致性**：
  - 神经网络结构与 DNQ 相同，依然是基础 DQN。
  - 动作解析逻辑中加入了障碍物判定，碰撞检测逻辑增强。

3.3 关键算法 DoubleDQN 设计

3.3.1 模型架构

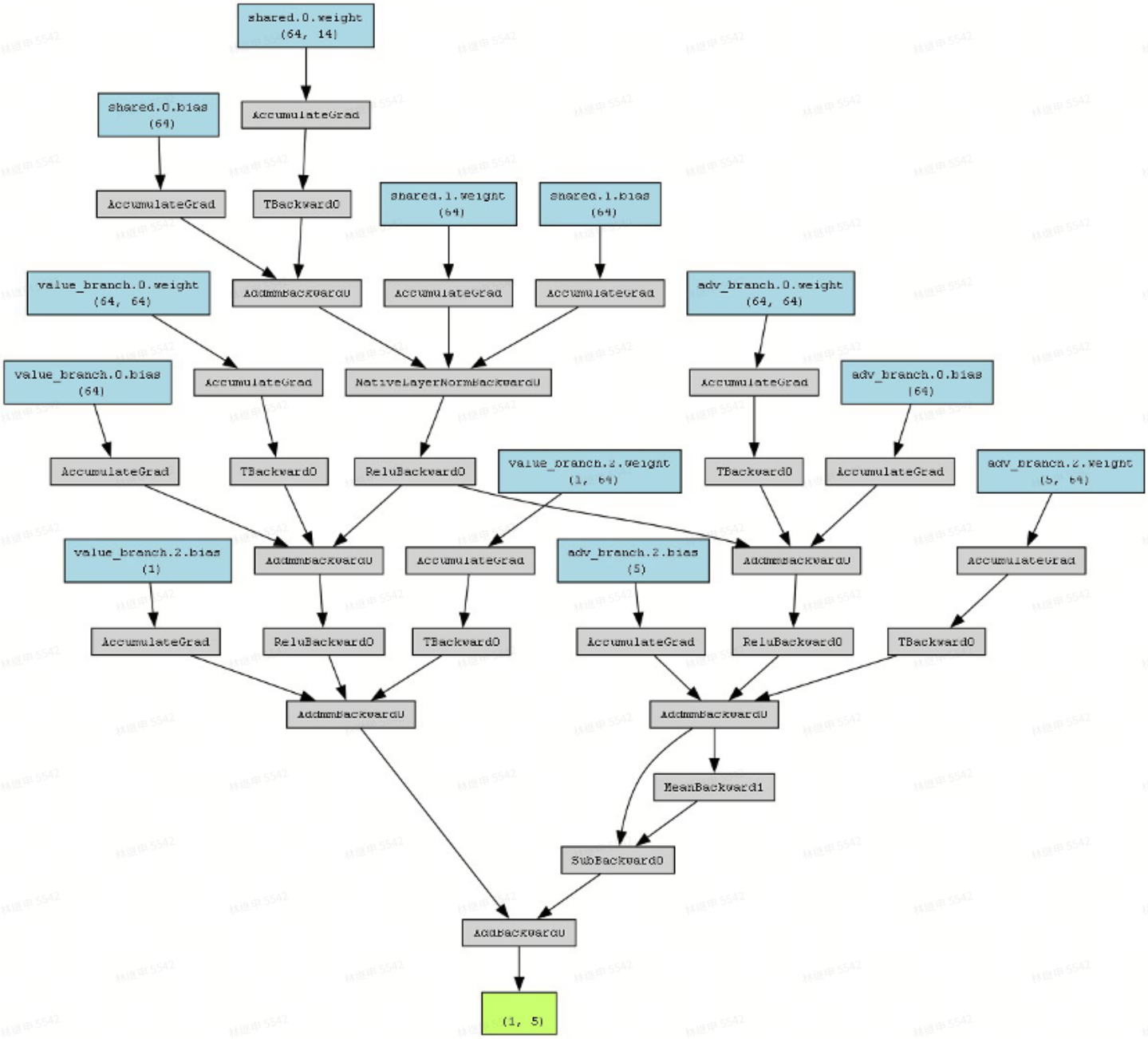

图 1 模型架构图

3.3.2 神经网络架构

采用的是 Dueling DQN 结构，每个智能体独立拥有一套网络。

- **共享隐藏层**：输入 (14 维) → Linear (64) → LayerNorm → ReLU
- **分支结构**：
  - Value 分支 / 状态价值 V(s) → Linear (64) → ReLU → Linear(1)
  - Advantage 分支 / 动作优势 A(s, a) → Linear (64) → ReLU → Linear (5)
- **合并 Q 值**：

$$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s,a') \right)$$

也即：

$$Q(s,a) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s,a';\theta,\alpha) \right)$$

这个 Q 值计算实现了对优势函数的零均值化，提高了学习稳定性。

3.3.3 目标值计算（Double DQN）

避免标准 DQN 的 Q 值过估计问题。

步骤：

- 当前策略网络（policy_net）选出最大 Q 值对应的动作：next_actions=policy_net(next_states).argmax(dim=1)
- 目标网络（target_net）用这个动作计算 Q 值：next_q=target_net(next_states).gather(1,next_actions)
- 目标 Q 值：targetQ=r+γ⊠(1-done)⊠Qtarget(s',a')

3.3.4 状态转换与观测

每次环境步进（step）后返回：

- 当前所有智能体的观测（位置、目标、障碍）
- 每个智能体的塑形奖励
- 是否终止（全部到达或步数上限）
- 碰撞次数（用于评估）

4. 实验过程

4.1 实验配置

该项目基于 Python 3.9，主要在 Windows 10 平台上开发和测试，并且在 Windows 11 平台上测试通过，使用的核心 Python 库包括：

库名称	用途说明
Numpy	数值计算，矩阵操作
Torch	深度学习框架，构建神经网络

Matplotlib	可视化，轨迹绘制与动画生成
collections.deque	经验回放缓冲区实现
Heapq	A* 算法中的优先队列（堆结构）
Random	初始化随机种子与随机位置
Time	训练时间记录

4.2 整体流程

4.2.1 创建环境与 Trainer

首先，构建多智能体交通路由仿真环境，并初始化强化学习训练器。

- **环境初始化**：

```
代码块
1  env = TrafficRoutingEnv(
2      grid_size=(GRID_ROWS, GRID_COLS),
3      num_agents=NUM_AGENTS,
4      obstacles=fixed_obstacles
5  )
```

- **状态空间维度**：共 14 维，由当前位置、目标位置（4 维）+ 邻居占用信息（9 维）+ 归一化曼哈顿距离（1 维）构成。
- **动作空间维度**：5 个离散动作（上、下、左、右、停留）。
- **训练器初始化**：

```
代码块
1  trainer = MADQNTrainer(
2      env=env,
3      num_agents=NUM_AGENTS,
4      state_dim=state_dim,
5      action_dim=action_dim
6  )
```

每个智能体独立维护一套 Dueling DQN 网络（策略网络+目标网络）、经验回放缓冲区及优化器。

4.2.2 训练阶段

通过强化学习迭代训练，让智能体学会在存在障碍物和其他智能体的动态环境中高效规划路径。

- **训练入口**：trainer.train(num_episodes=NUM_EPISODES)
- **每个 Episode 的主要流程**：
  - 调用 env.reset()，智能体恢复到构造时固定的起点和目标。
  - 进入交互循环：
    - 各智能体依据 ε-贪心策略选择动作。
    - 调用 env.step(actions) 更新环境状态并返回奖励。
    - 将交互产生的 (state, action, reward, next_state, done) 五元组存入经验缓冲区。
    - 调用 optimize_agent() 基于经验回放执行 DQN 更新。
  - 若所有智能体到达目标或步数达到上限（50 步），该回合终止。
  - 执行：
    - ε 递减（平衡探索与利用）。
    - 每间隔 target_update 轮，将策略网络的参数同步到目标网络。
- **训练结束产出**：episode_returns 记录每个回合的累积奖励，用于绘制训练收敛曲线。

4.2.3 可视化阶段

在训练完成后，通过贪心策略（ε = 0）执行一次完整的仿真回合，展示智能体的实际运动路径。

- **可视化入口**：animate(trainer, env)
- **主要流程**：
  - 再次调用 env.reset()，智能体返回初始起点。
  - 所有智能体基于训练好的策略执行动作（不再包含随机探索）。
  - 动态记录每一步所有智能体的位置。
  - 绘制网格地图，展示。
- **结果呈现**：
  - 以动态图的形式展示智能体如何协同避障、避免碰撞并顺利到达各自目标。
  - 保存为 gif 文件，便于汇报与演示。
  - 绘制训练收敛曲线，以 png 保存。

5. 实验结果与分析

5.1 轨迹动态演示

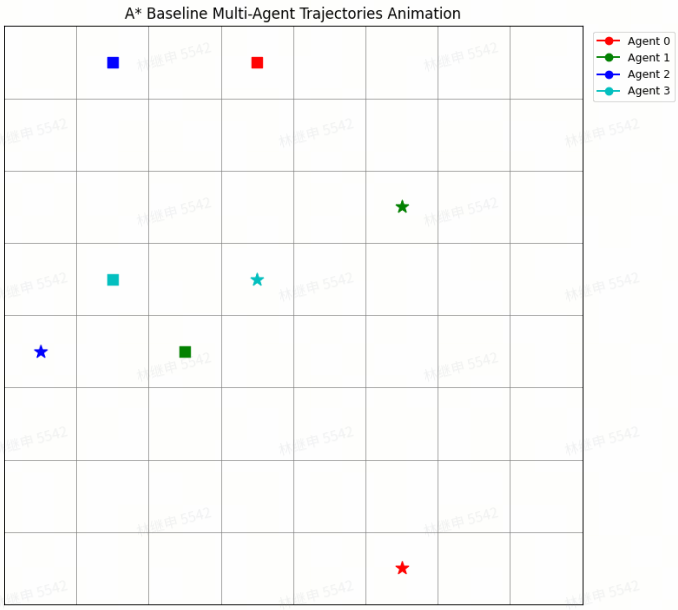

图 2 A* 算法（基准）

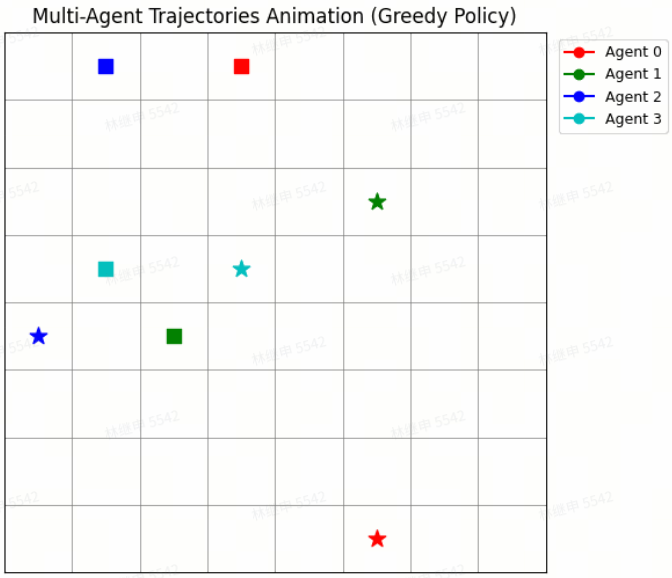

图 3 DQN 算法

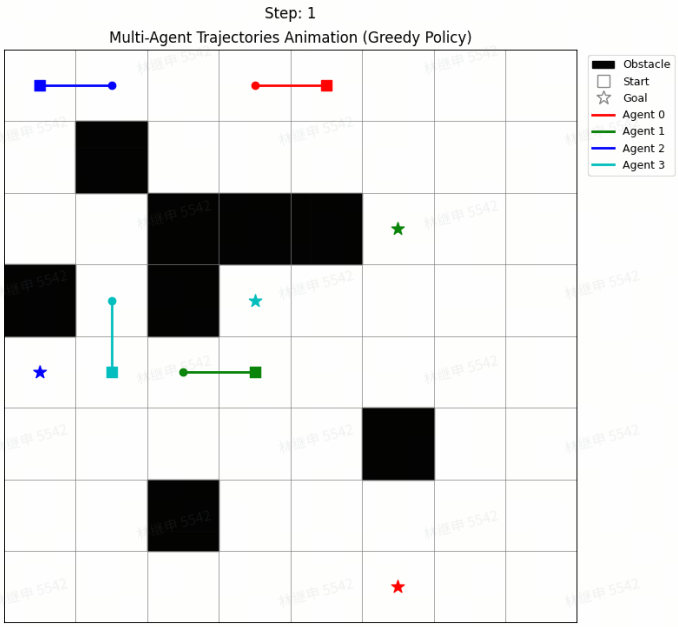

图 4 DQN 算法 + 静态障碍

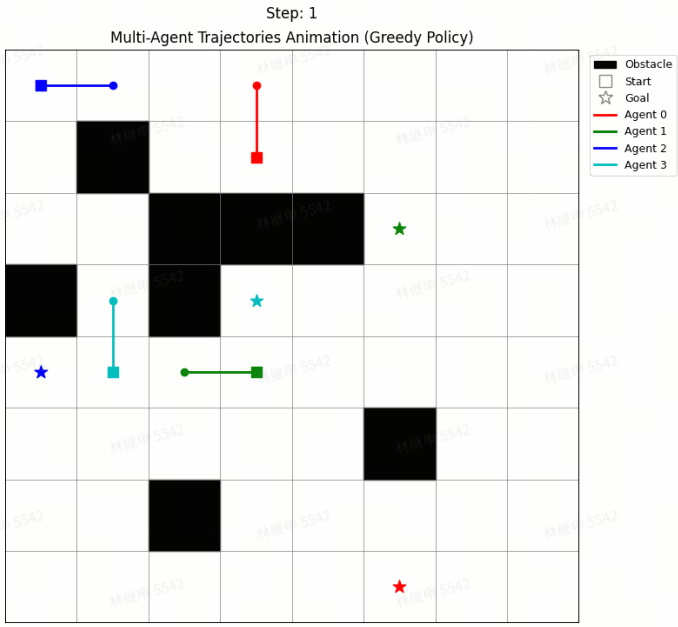

图 5 DoubleDQN 算法

5.2 训练结果展示

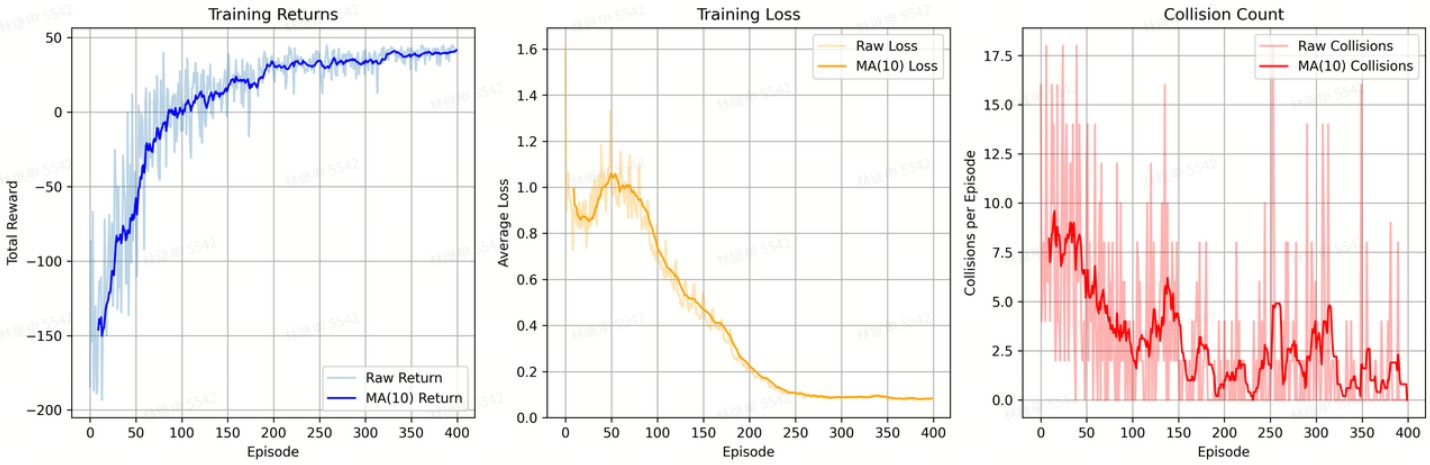

图 6 DQN 训练结果曲线

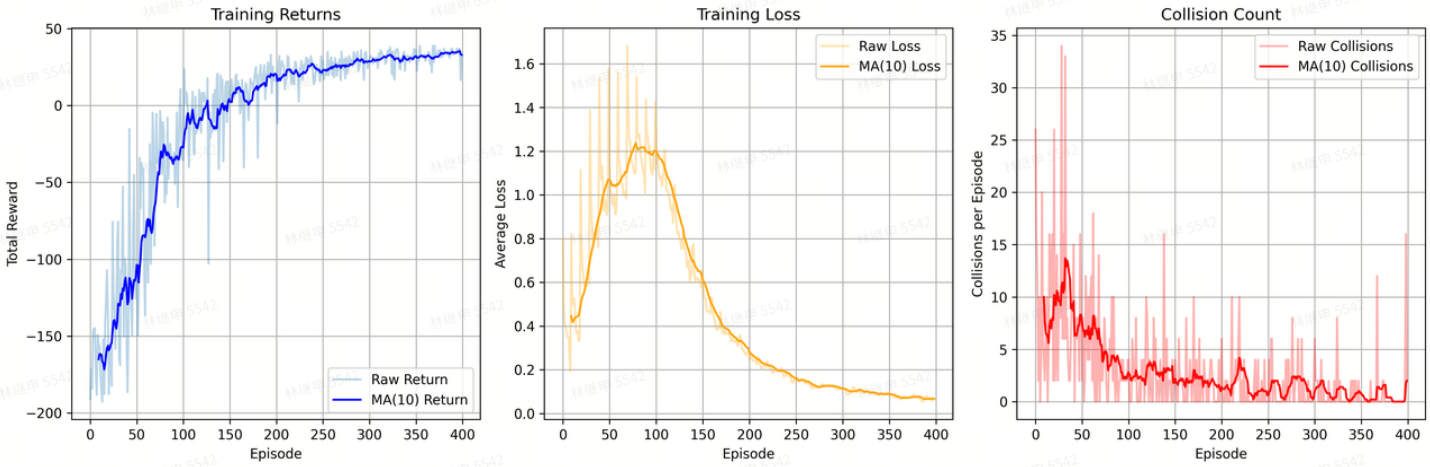

图 7 DQNObstacle 训练结果曲线

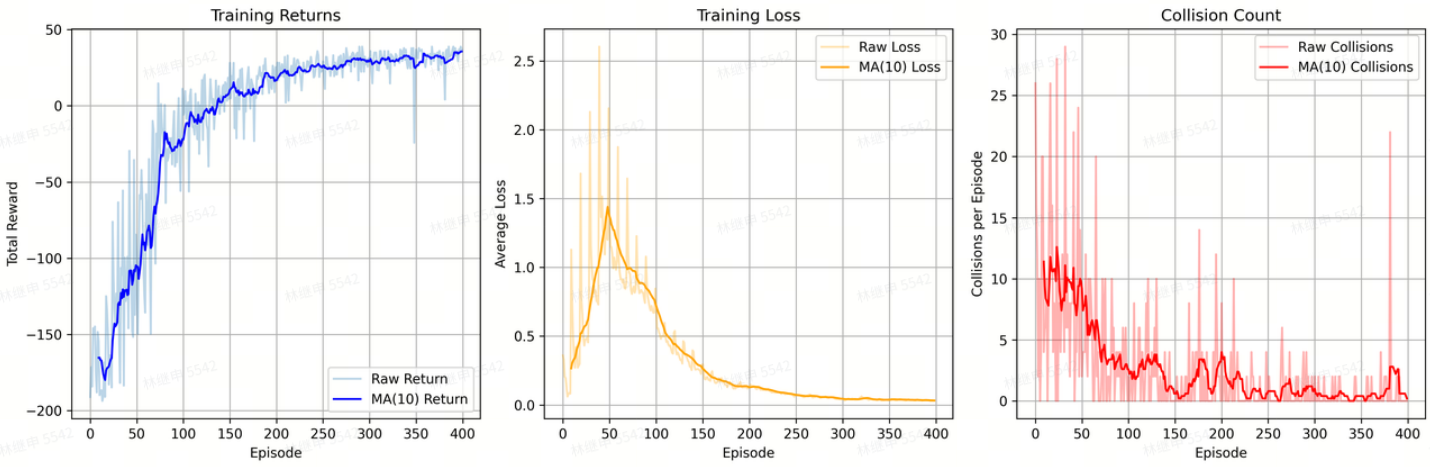

图 8 DoubleDQN 训练结果曲线

5.3 实验结果分析

5.3.1 A* 基线表现

- **优点：**
  - 无需训练，路径规划快速。
- **缺点：**

- 无动态避障机制。
- 无法处理多智能体的碰撞（尤其是交换位置碰撞和同格碰撞）。
- 在高密度环境中容易出现死锁和路径交叉，导致系统整体性能较差。

5.3.2 DQN（无障碍）表现

- **优点**：
  - 在无障碍环境中可以学会有效的避让策略。
  - 收敛速度中等（约 300 轮）。
- **缺点**：
  - 奖励曲线波动大，存在明显震荡。
  - 损失函数长期保持在较高水平，训练不够平滑。
  - 在密集智能体环境下，碰撞率波动大，偶尔存在严重的拥堵。

5.3.3 DQN-Obstacle（有障碍）表现

- **优点**：
  - 增加了障碍物感知，具备一定避障能力。
- **缺点**：
  - Reward 上升速度慢，长期处于中低水平。
  - 碰撞次数持续较高，且训练后期稳定性不足，环境复杂度显著影响性能。

5.3.4 Double DQN（Dueling 结构+Double Q 目标）表现

- **优点**：
  - 收敛速度快。
  - 损失函数在 150 轮后迅速下降至 0.5 以下，整体下降趋势平滑。
  - Reward 曲线平滑上升，最终接近理论最优（约 40）。
  - 碰撞次数持续下降，后期趋近于 0，具有良好的路径协调与避让能力。
  - 高稳定性，训练后期无明显波动，适用于复杂障碍场景。
- **缺点**：
  - 相较于 DQN，训练投入时间更长。
  - 模型泛化能力依赖于障碍物与起终点的随机分布，对于变化过大的环境仍需微调。

6. 总结

6.1 优势与不足（针对 Double DQN）

6.1.1 主要优势

- **强鲁棒性**：在障碍物和多智能体环境下依旧能保持高效避障和低碰撞。
- **收敛速度快**：明显优于标准 DQN 和 DQN-Obstacle。
- **高稳定性**：训练过程平滑，训练后期 Reward 与 Loss 曲线无剧烈震荡。
- **极佳的碰撞控制能力**：最终碰撞率趋近于 0，且较为稳定，路径安排协调。
- **优化有效**：Dueling 结构强化了状态价值感知，Double DQN 有效减少了 Q 值过估计。

6.1.2 存在的不足

- **训练成本较高**：相比 DQN 需要更大的训练投入。
- **泛化有限**：在障碍极度变化或者大规模环境扩展下，仍然需要微调网络结构或增加训练数据。
- **无全局最优保证**：基于 Q-Learning 的局部更新策略，本质上无法保证全局最优解。

6.1.3 对比其他方法

- A* 适用于简单场景，但不具备多智能体协同能力。
- DQN 与 DQN-Obstacle 能够解决部分路径冲突，但在复杂环境中表现不稳定。
- Double DQN 是当前方案中表现最优的模型，兼顾了高效避障、快速收敛和低碰撞率，适用于需要动态路径调整与高智能体密度的工业场景。

6.2 项目核心价值

本项目通过实现基础 Double DQN 算法，并结合曼哈顿距离潜力塑形技术，对多智能体路径规划问题进行了探索和改进。项目的核心价值主要体现在以下几个方面：

- **算法实践与对比**：项目实现了基础 DQN 和 Double DQN 算法，通过对比两者的性能，帮助我们更好地理解不同算法在多智能体路径规划中的表现。特别是 Double DQN 通过减少 Q 值过估计问题，显著提升了路径规划的效率和稳定性。
- **奖励塑形的应用**：引入曼哈顿距离潜力塑形项，加速了训练的收敛速度。这一改进使得智能体能够更快地学习到有效的路径规划策略，减少了训练时间。
- **碰撞和拥堵的减少**：通过改进算法和奖励机制，项目有效减少了智能体之间的碰撞和拥堵现象。这不仅提高了路径规划的成功率，还提升了系统的整体效率。
- **可视化展示**：项目通过可视化技术展示了智能体的移动轨迹和策略演变过程。这使得我们能够直观地观察智能体的行为模式，验证算法的有效性，并为进一步的改进提供了依据。

6.3 应用领域展望

6.3.1 智慧仓储与物流调度

在大型自动化仓储中心，数百台自动引导车（AGV）需要在有限的空间内高效协作，完成货物的搬运和存储任务。这些 AGV 需要在复杂的环境中动态规划路径，避免相互碰撞和堵塞，同时优化路径以提高整体物流效率。

6.3.2 智能交通系统

城市道路交通中，通过智能信号灯控制和动态路线推荐，优化交通流量，减少拥堵。智能交通系统需要实时分析交通状况，动态调整信号灯时长和推荐最优路线，以应对突发的交通事件和拥堵。

6.3.3 无人机编队与机器人集群

在无人机编队飞行、巡检任务或机器人集群作战中，多个智能体需要协同行动，动态调整路径以避免碰撞。这些场景要求智能体能够快速响应环境变化，实时调整策略。

6.3.4 生产制造车间

在现代化工厂的生产线上，多个机械臂或运输机器人需要协同工作，完成复杂的生产任务。这些机器人需要在动态环境中高效搬运物料，避免相互干扰，提高生产效率。