# A*-GUIDED DEEP REINFORCEMENT LEARNING FOR SINGLE- AND DUAL-AGENT NAVIGATION: CROSS-ALGORITHM EVALUATION AND TRANSFORMER-BASED POLICIES

A thesis presented in partial fulfilment of the requirements for the degree of
Master of Information Science
in
Computer Science
at Massey University, Auckland,
New Zealand.

Yahong Liu

2026

# Contents

# List of Tables

# List of Figures

# List of Abbreviations

| | |
|---|---|
| **A*** | A-Star Search Algorithm |
| **A2C** | Advantage Actor-Critic |
| **A3C** | Asynchronous Advantage Actor-Critic |
| **AGV** | Automated Guided Vehicle |
| **BFS** | Breadth-First Search |
| **BenchMARL** | Benchmarking Multi-Agent Reinforcement Learning |
| **CNN** | Convolutional Neural Network |
| **COMA** | Counterfactual Multi-Agent Policy Gradients |
| **CTDE** | Centralized Training with Decentralized Execution |
| **CV** | Coefficient of Variation |
| **D*** | Dynamic A-Star |
| **DDPG** | Deep Deterministic Policy Gradient |
| **DFS** | Depth-First Search |
| **DPG** | Deterministic Policy Gradient |
| **DQN** | Deep Q-Network |
| **DRL** | Deep Reinforcement Learning |
| **DWA** | Dynamic Window Approach |
| **FFN** | Feed-Forward Network |
| **GAE** | Generalized Advantage Estimation |
| **GPU** | Graphics Processing Unit |
| **IDA*** | Iterative Deepening A-Star |
| **IQL** | Independent Q-Learning |
| **KL** | Kullback-Leibler Divergence |
| **LiDAR** | Light Detection and Ranging |
| **LPA*** | Lifelong Planning A-Star |
| **MADDPG** | Multi-Agent Deep Deterministic Policy Gradient |
| **MAPF** | Multi-Agent Path Finding |
| **MAPPO** | Multi-Agent Proximal Policy Optimization |
| **MARL** | Multi-Agent Reinforcement Learning |

| | |
|---|---|
| **MCTS** | Monte Carlo Tree Search |
| **MDP** | Markov Decision Process |
| **MLP** | Multilayer Perceptron |
| **PPO** | Proximal Policy Optimization |
| **QMIX** | Q-value Mixing Network |
| **RL** | Reinforcement Learning |
| **RLHF** | Reinforcement Learning from Human Feedback |
| **SAC** | Soft Actor-Critic |
| **SGD** | Stochastic Gradient Descent |
| **TD** | Temporal Difference |
| **TD3** | Twin Delayed Deep Deterministic Policy Gradient |
| **TRPO** | Trust Region Policy Optimization |
| **UAV** | Unmanned Aerial Vehicle |
| **UCB** | Upper Confidence Bound |
| **UCS** | Uniform Cost Search |
| **VDN** | Value Decomposition Network |
| **VMAS** | Vectorized Multi-Agent Simulator |
| **WAT/TA-WAT** | Waypoint-Aware Transformer (also referred to as Task-Adaptive Waypoint-Aware Transformer) |

# Abstract

Integrating classical path planning with deep reinforcement learning (DRL) offers a promising approach for navigation in obstacle-dense environments, where purely learned policies often suffer from sparse rewards, inefficient exploration, unstable convergence, and collision-prone behaviour. This thesis investigates the integration of the A-star (A*) path-planning algorithm with deep and multi-agent reinforcement learning (MARL), with a particular focus on reproducibility, cross-algorithm evaluation, policy architecture design, and the transition from single-agent to dual-agent navigation. The study builds on previous A*-guided DRL research by reproducing and extending its core ideas within the Vectorized Multi-Agent Simulator (VMAS) and Benchmarking Multi-Agent Reinforcement Learning (BenchMARL) frameworks.

The thesis makes four main contributions. First, it establishes a reproducible experimental pipeline for evaluating A*-guided reinforcement learning in navigation tasks. Second, it examines how A*-generated waypoint guidance affects different reinforcement learning algorithms, including Multi-Agent Proximal Policy Optimization (MAPPO), Multi-Agent Deep Deterministic Policy Gradient (MADDPG), and the Q-value Mixing Network (QMIX). Third, it evaluates the use of structured waypoint information through a Task-Adaptive Waypoint-Aware Transformer configuration. Fourth, it investigates how A*-guided learning and waypoint-aware policies behave when extended from controlled single-agent navigation to dual-agent navigation.

In the proposed framework, A* is used to generate waypoint paths that support reinforcement learning through reward shaping and, in some configurations, through explicit waypoint observation features. For the multilayer perceptron (MLP) baseline and Standard Transformer configurations, A*-derived guidance is used during training and removed during evaluation. For the Task-Adaptive Waypoint-Aware Transformer configurations, A*-based reward shaping is disabled during evaluation, but A*-derived waypoint observation features may remain part of the policy input. The baseline reproduction shows positive descriptive differences in final-window evaluation reward and the collision-penalty proxy in controlled single-agent settings, particularly in compact obstacle-dense environments. However, the results also show that A* guidance does not necessarily improve sample efficiency or convergence speed and introduces additional

computational overhead. Extending the framework to MAPPO, MADDPG, and QMIX reveals that the effects of A* guidance are algorithm-dependent, with different learning paradigms responding differently to planning-derived reward signals.

The architectural evaluation shows that a Standard Transformer Policy Baseline does not automatically outperform the Fully Connected MLP Policy Baseline. This indicates that simply replacing an MLP with a Transformer is insufficient for improving navigation performance. In contrast, the Task-Adaptive Waypoint-Aware Transformer shows positive descriptive trends in selected final performance metrics, including higher average-reward and final-window evaluation-reward values in some configurations. However, the five-seed statistical analysis does not establish statistically significant improvements after Holm-Bonferroni correction.

The dual-agent experiments reveal an important limitation of directly transferring single-agent A*-guided strategies to multi-agent navigation. While A* guidance reduces collisions in single-agent settings, independently generated A* paths can create inter-agent path conflicts in dual-agent scenarios. This produces a qualitative shift in the role of A* guidance: its collision-reduction benefits do not automatically transfer to multi-agent settings, although it may still support generalization and task completion in some cases. The results also show that scaling waypoint-aware Transformer policies to dual-agent navigation increases training difficulty and produces algorithm-dependent effects. MADDPG benefits most clearly from the waypoint-aware architecture, QMIX shows mixed effects, and MAPPO exhibits instability under the tested configuration.

Overall, this thesis demonstrates that classical planning signals can provide useful structure for reinforcement learning-based navigation, but their benefits depend strongly on the learning algorithm, policy architecture, evaluation setting, and number of agents. The findings clarify both the potential and the limitations of A*-guided DRL and MARL. They suggest that waypoint-aware Transformer policies are a promising direction for structured navigation learning, while also highlighting the need for larger-scale statistical validation, improved multi-agent coordination mechanisms, and more robust approaches for resolving conflicts between independently planned paths.

# Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Napoleon Reyes, for his invaluable guidance, continuous encouragement, and constructive support throughout the course of this research. I am also deeply thankful to my family for their unwavering support and understanding, which gave me the strength to complete this work.

# Chapter 1

# Introduction

## 1.1 Background and Motivation

Reinforcement learning (RL) is an important field of machine learning in which an agent interacts with the environment and learns an optimal policy through reward signals [2]. RL has made significant contributions across many domains, including AlphaGo in the game of Go [5] and Atari video games [6]. More recently, Reinforcement Learning from Human Feedback (RLHF) has become an important technique for aligning large language models and generative artificial intelligence systems with human preferences, further demonstrating the broad importance of RL in modern artificial intelligence [7]. With the emergence of deep neural networks, reinforcement learning has evolved into deep reinforcement learning (DRL), enabling agents to operate directly on high-dimensional state spaces and learn complex decision-making policies. DRL algorithms represented by deep Q-Networks (DQN) emerged [8]. DRL has also achieved strong results in robotics, including end-to-end learning of visual motion strategies [9].

However, traditional single-agent reinforcement learning faces certain limitations when scaling to complex tasks. First, the exploration efficiency of a single agent is relatively low, often requiring a large number of trial-and-error interactions to discover effective policies. This may lead to slow convergence and the risk of being trapped in local optima [8, 10]. Second, training single-agent models usually requires substantial computational and time resources. Especially in high-dimensional environments, this resource consumption increases significantly [6, 8].

As the modern world grows more complex, single-agent reinforcement learning has become insufficient for many emerging applications. Research has expanded to multi-agent systems and real-world applications. For example, RoboBallet [11], released in 2025, enabled multiple robotic arms to collaborate efficiently using reinforcement learning. This improved task completion in industrial production and highlighted RL's

potential in robot coordination and motion planning. Such results underscore the growing importance of multi-agent reinforcement learning (MARL) for real-world robotic coordination and motion planning problems.

Nevertheless, traditional single-agent research faces multiple challenges when extended to multi-agent learning. First, with multiple agents learning simultaneously, the environment becomes dynamic and non-stationary from each agent's perspective, making convergence more difficult [2, 12]. Second, in multi-agent learning, agents are typically trained independently. This raises the challenge of determining how agents can effectively communicate and collaborate, and of identifying mechanisms that achieve optimal system performance. Finally, as the number of agents increases, the number of state-action pairings grows exponentially, leading to scalability problems [2]. Thus, when single-agent methods are used in multi-agent applications, the dynamic environment, increased need for coordination, and scalability issues all become significant challenges. As a result, multi-agent algorithms have emerged.

MARL algorithms are generally divided into three categories: value-based, policy-based, and hybrid methods. First, value-based methods, such as the Value Decomposition Network (VDN) [13] and Q-value Mixing Network (QMIX) [14], decompose the joint value function to enable decentralized execution. In contrast, policy-based methods, such as Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [15] and Multi-Agent Proximal Policy Optimization (MAPPO) [16], achieve multi-agent collaboration by using a centralized critic to guide the decentralized training of each agent's actor. Finally, hybrid methods, such as Counterfactual Multi-Agent Policy Gradients (COMA) and actor-critic variants [17], combine value optimization and policy optimization to improve stability and sample efficiency. While this thesis is motivated by challenges in multi-agent navigation, single-agent settings are intentionally employed as controlled experimental probes, followed by a limited extension to a dual-agent setting. The experiments do not evaluate generalisation to teams of three or more agents.

## 1.2 Statement of the problem

In complex and dense environments, traditional path planning algorithms such as the A-star (A*) algorithm have proven effective in generating efficient and collision-free paths [18]. Building upon this, incremental planning algorithms such as Lifelong Planning A-star (LPA*) and Dynamic A-star Lite (D* Lite) were introduced to update paths in response to local map changes or partially unknown areas [19, 20]. Although these algorithms can adapt to local unknown areas and local map changes, they remain fundamentally heuristic-based and reactive planners. Specifically, they rely entirely on known map information and heuristic functions, lacking the ability to learn environmental structures from past experiences. As a result, when the environment undergoes

large-scale, high-frequency changes, even D* Lite may suffer from significant computational overhead during replanning.

In contrast, Deep Reinforcement Learning (DRL) enables near-real-time, end-to-end decision-making via policy networks, demonstrating strong potential for partially observable and complex state spaces [6]. However, DRL training typically suffers from low sample efficiency, instability, high sensitivity to hyperparameters, and insufficient exploration, which can easily lead the agent to become trapped in local optima and to experience difficulty converging [2].

These observations suggest that traditional planning and DRL are complementary: traditional methods provide deterministically optimal solutions in static environments, while DRL enables continuous optimization under uncertainty. Building on this cohesion, the present study investigates how deterministic solutions from traditional algorithms (e.g., A*) can help DRL agents explore large and uncertain state spaces more effectively. Recent research shows that integrating a classical pathfinding algorithm like A* with deep reinforcement learning accelerates learning, improves policy stability, and enhances decision-making efficiency, particularly in environments with many obstacles [4].

Despite these advances, existing studies are still limited to single-agent scenarios and relatively simple obstacle configurations. However, little attention has been given to maze-like environments or to multi-agent coordination—both of which are critical for cooperative navigation tasks. Furthermore, while transformer-based architectures have recently shown promise in single-agent reinforcement learning for capturing long-term dependencies and structured information [21, 22, 23], existing works have not systematically explored how deterministic planning signals from classical algorithms like A* can be leveraged to train transformer-based DRL agents.

In response, this study investigates how deterministic planning signals from classical algorithms such as A* can be integrated into deep reinforcement learning. It compares the resulting learning behaviour across three reinforcement learning algorithms in controlled single-agent settings and subsequently conducts a limited dual-agent extension to examine initial coordination-related effects. The study does not evaluate larger multi-agent teams.

## 1.3 Objectives and Aims

The overall objective of this thesis is to integrate traditional path-planning methods with deep reinforcement learning (DRL) using the Vectorized Multi-Agent Simulator (VMAS) simulation framework and the Benchmarking Multi-Agent Reinforcement Learning (BenchMARL) training pipeline. The goal is to examine the interaction between these approaches under controlled navigation conditions. In particular, the study

combines A* guidance with DRL and compares three MARL algorithms within this framework. Although MARL algorithms are employed, the primary experiments use controlled single-agent navigation to isolate the effects of planning guidance, followed by a limited dual-agent extension. No experiments with three or more agents were conducted.

This thesis is carried out in four stages:

1. **Reproduction of the baseline study:** First, a relevant A*-guided reinforcement learning study from the literature is examined and reproduced using the VMAS simulation framework and the BenchMARL training pipeline. This stage establishes a controlled baseline for examining A* planning within the reinforcement learning process; it does not by itself establish general effectiveness beyond the tested conditions.

2. **Extension to multiple reinforcement learning algorithms:** Building on the reproduced framework, A*-guided learning is applied to MAPPO, MADDPG, and QMIX. This stage compares how waypoint guidance behaves across the three algorithmic paradigms under the tested conditions.

3. **Architectural evaluation of waypoint-aware Transformer policies:** To address the limitations of the fully connected multilayer perceptron (MLP) Policy Baseline, Transformer-based policy architectures are introduced. In particular, a Task-Adaptive Waypoint-Aware Transformer (TA-WAT) configuration is designed and evaluated by incorporating A*-derived waypoint observations into the tested Transformer policy.

4. **Extension to dual-agent navigation:** Finally, the A*-guided learning framework and waypoint-aware Transformer policies are extended from controlled single-agent navigation tasks to dual-agent navigation scenarios. This stage examines how the tested configurations behave when moving from one agent to two agents and identifies limitations that arise in the dual-agent setting. It does not evaluate scalability or robustness for larger agent populations.

### 1.3.1 Research Questions

- RQ1: How can A*-generated trajectories be effectively integrated into deep reinforcement learning to accelerate convergence, reduce exploration randomness, and improve policy stability?

- RQ2: How well does the A*-augmented learning framework generalize across different multi-agent reinforcement learning algorithms when applied to single-agent navigation tasks?

- RQ3: In the context of integrating path planning and deep reinforcement learning, can introducing a Transformer architecture with a self-attention mechanism to replace the traditional MLP improve task-completion proxies, convergence performance, and overall policy quality?
- RQ4: Can waypoint-aware Transformer-based Deep Reinforcement Learning architectures improve navigation performance by using A*-generated waypoints as structured guidance?
- RQ5: Are the training strategies used in single-agent A*-guided Deep Reinforcement Learning applicable to dual-agent A*-guided Deep Reinforcement Learning, and how does the transition from single-agent to dual-agent navigation affect learning performance, collision avoidance, and generalization?

### 1.3.2 Hypotheses

- H1: Incorporating A*-generated trajectories into deep reinforcement learning improves sample efficiency and convergence speed by providing structured intermediate supervision, while also reducing the collision-penalty proxy and improving policy stability in single-agent navigation tasks.
- H2: A* guidance consistently reduces collisions in single-agent scenarios across MARL algorithms, as waypoint information resolves credit assignment regardless of the learning paradigm. However, the magnitude of improvement varies across algorithms, with continuous-action algorithms (MAPPO, MADDPG) more precisely following A* waypoints than discrete-action algorithms (QMIX), and off-policy methods benefiting more due to experience replay.
- H3: The Standard Transformer Policy Baseline is compared with the Fully Connected MLP Policy Baseline under A*-guided reinforcement learning to determine whether replacing the MLP with a Transformer improves navigation performance.
- H4: Under otherwise identical Transformer architectures and training settings, adding A*-derived waypoint observations to the policy input may produce descriptive differences in navigation performance relative to the Standard Transformer baseline. The experiment evaluates the complete waypoint-input condition and does not isolate the effects of self-attention, relative positional encoding, gated feed-forward networks, or local attention.
- H5: Training strategies developed for single-agent A*-guided reinforcement learning may not transfer directly to the tested dual-agent setting. Independently generated A* paths may introduce inter-agent conflicts, and their effects on collision-related proxies, learning stability, and evaluation reward are assessed only for two

agents.

## 1.4 Research Contributions

This thesis makes four main contributions to the study of planning-guided reinforcement learning for navigation. These contributions concern experimental reproducibility, controlled cross-algorithm evaluation, explicit waypoint-informed policy inputs, and the extension from single-agent to dual-agent navigation.

1. **A reproducible evaluation framework for A*-guided reinforcement learning.** The study reproduces and extends an existing A*-guided reinforcement learning approach within VMAS and BenchMARL. It establishes a unified pipeline for comparing planner-derived reward shaping and policy learning under controlled navigation conditions, while distinguishing the A*-derived information available during training and evaluation.

2. **A controlled cross-algorithm evaluation of A*-guided learning.** The same planning-guidance framework is evaluated with MAPPO, MADDPG, and QMIX under consistent environment and evaluation settings. This comparison shows that the observed effects of A* guidance are algorithm-dependent rather than uniform across learning paradigms.

3. **An evaluation of explicit A*-derived waypoint observations in Transformer policies.** The study compares a Standard Transformer with an otherwise identical waypoint-aware Transformer. Both variants use the same input dimension, network structure, relative positional encoding, gated feed-forward network, local attention, and training settings; their only intentional model-input difference is whether the shared five-dimensional waypoint field is zero-filled or contains A*-derived waypoint values. The comparison therefore evaluates the complete waypoint-input condition and does not establish separate effects for the shared Transformer components.

4. **An extension from single-agent to dual-agent navigation.** The A*-guided learning framework and waypoint-aware policy configurations are extended to a dual-agent scenario. The resulting analysis identifies an important limitation of direct transfer: independently generated A* paths can create inter-agent path conflicts, and the observed effects vary across algorithms and evaluation measures. This contribution is limited to the transition from one agent to two agents and does not establish scalability to larger teams.

## 1.5 Significance of the Study

### 1.5.1 Academic Significance

Existing research on navigation tasks primarily focuses on single-agent settings. Although some recent studies have attempted to integrate A* planning with deep reinforcement learning in complex environments [4], these efforts are mostly constrained to single-agent formulations or highly specific scenarios. The use of A* as a heuristic mechanism in cooperative multi-agent navigation remains comparatively underexplored. This thesis addresses only an initial part of that broader gap through a controlled dual-agent experiment and does not evaluate larger cooperative teams.

This work addresses the unexplored gap by:

1. Reproducing and validating the original method within the VMAS simulation framework and the BenchMARL training pipeline, establishing a unified experimental setup that integrates the classical A* pathfinding algorithm with reinforcement learning. This framework provides a reproducible foundation for analyzing the influence of planner-derived information on the learning process.

2. Conducting a systematic evaluation of A*-generated waypoint guidance across multiple reinforcement learning algorithms, including MAPPO, MADDPG, and QMIX. This analysis examines how planning-derived information affects learning efficiency, stability, and navigation performance under a consistent experimental setting.

3. Incorporating Transformer-based policy architectures into the learning framework and systematically evaluating whether structured waypoint information can be utilized more effectively than in conventional MLP-based policies. In particular, a Task-Adaptive Waypoint-Aware Transformer (TA-WAT) configuration is designed and evaluated by incorporating A*-derived waypoint observations into the tested Transformer policy. The contribution concerns the design and evaluation of this waypoint-input configuration rather than the proposal of a fundamentally new Transformer architecture.

4. Extending both A*-guided learning and waypoint-aware Transformer policies from controlled single-agent navigation tasks to a dual-agent navigation scenario. This extension provides a comparison between one-agent and two-agent conditions and identifies limitations observed with two agents. Scalability and robustness beyond two agents remain untested.

The experimental results show positive descriptive differences for A*-guided reward shaping compared with DRL trained without A* guidance. In the reported single-agent

comparisons, the final-window evaluation-reward value is up to 54.9% higher and the collision-penalty proxy is up to 40.0% lower in compact environments.

### 1.5.2 Practical Significance

The tested framework may inform future research on cooperative navigation, including multi-Unmanned Aerial Vehicle (multi-UAV) delivery, automated warehouse management, and fleet coordination. Such applications require coordination and path planning among larger teams in dynamic environments. However, the evidence in this thesis is limited to simulated single-agent and dual-agent maze navigation. It does not demonstrate deployment readiness, transferability to real-world systems, or scalability to larger teams. Existing research indicates that hybrid approaches combining heuristic search and DRL may be relevant to multi-UAV formation control [24], but the application of the present framework to intelligent transportation or smart manufacturing remains a direction for future investigation [25].

## 1.6 Scope and Limitations

This study focuses primarily on controlled single-agent navigation tasks built on the VMAS and BenchMARL platforms and includes a limited dual-agent extension. It compares pure DRL (DRL-only) and A*+DRL hybrid (Hybrid) strategies using MAPPO as the primary algorithm, with QMIX and MADDPG included for comparison. Transformer policies are evaluated using reward, task-completion, collision-penalty, and stability measures. No experiments involving three or more agents were conducted.

Nonetheless, the study has several limitations:

1. Platform Constraints: Experiments were conducted exclusively on VMAS and BenchMARL. While these platforms ensure reproducibility, results may not fully generalize to other simulation environments or real-world robotic systems.

2. Reward Function Generality: The reward function was directly adopted from prior literature[4]. In actual problem solving, this study made appropriate improvements and optimizations. It did not conduct systematic comparative experiments to evaluate the performance differences between different reward designs. Therefore, the universality and transferability of the reward function in different scenarios still need further verification.

3. Algorithm Coverage: Although representative MARL algorithms (MAPPO, QMIX, MADDPG) were included, other emerging methods were not explored deeply.

4. Experimental Scale: Multi-agent evaluation was limited to two agents. Consequently, the study cannot establish scalability, coordination effectiveness, or robustness for teams of three or more agents. Larger-team experiments were not conducted because of time and computational-resource constraints.

In summary, this study compares DRL-only and Hybrid strategies within specific environments and algorithmic boundaries. Generalisation to other environments and scalability beyond two agents remain untested and require future evaluation.

## 1.7 Thesis Structure Overview

Chapter 1: Introduction

This chapter discusses the motivation for merging traditional path-planning algorithms with deep reinforcement learning, which lies at the very heart of this thesis. It introduces VMAS, BenchMARL, the experimental platforms, the research hypotheses being carried out, and the constraints to be understood in the study. A broad outline of the paper's organization is provided.

Chapter 2: Literature Review

This chapter provides an ordered review of relevant literature on the historical development of classical pathfinding algorithms, the foundations and architectural choices in deep reinforcement learning, and previous attempts to integrate DRL with traditional planning. In addition, it provides a comprehensive survey of mainstream multi-agent reinforcement learning algorithms, which provides an adequate theoretical background for the methodology and experiments of this research. It also analyzed the current research gaps.

Chapter 3: Methodology

This chapter introduces the methodological framework used in this study, including the definition of the environment, the integration of the A* algorithm and deep reinforcement learning, the design of the reward function, the algorithm design, and the overall experimental process.

Chapter 4: Experimental Development and Design Decisions

This chapter details the experimental development process. It therefore discusses environment setup, and implementation of both A* and DRL methods on VMAS + BenchMARL as well as Optuna for hyperparameter tuning attempts at reward-function design use of different MARL algorithms, and finally memory-based models including Transformer-enhanced agents. The major steps involved in this study are discussed herein, along with insights gained at different stages of the research.

Chapter 5: Baseline Reproduction and Validation

This chapter reproduces the A*-DRL integration approach proposed by Lopez-Yepez et al. under a single-agent setting implemented within the multi-agent frameworks VMAS and BenchMARL. This reproduction establishes a benchmark and checks the experimental pipeline for subsequent investigations.

Chapter 6: Cross-Algorithm Evaluation of A*-Guided Learning

This chapter extends the baseline evaluation by examining the A*-DRL hybrid approach under representative multi-agent reinforcement learning algorithms, including MAPPO, QMIX, and MADDPG. Although the environment remains a single-agent navigation task, these algorithms are applied in their degenerated single-agent formulations to enable controlled and fair comparisons across different algorithmic paradigms.

Chapter 7: Evaluation of Task-Adaptive Waypoint-Aware Transformer (TA-WAT) Policies

By comparing MAPPO policies implemented with MLP and Transformer networks, this chapter evaluates descriptive differences between the complete policy architectures. It subsequently compares the Standard Transformer with an otherwise identical waypoint-aware Transformer that additionally receives A*-derived waypoint observations. These comparisons do not isolate the effect of self-attention or any individual Transformer component.

Chapter 8: Feasibility and Limits of A* Guidance in Dual-Agent Navigation

Building upon the controlled single-agent evaluation and algorithm-level extensions, this chapter explores the A*-guided approach in a dual-agent environment, aiming to examine its observed behaviour and limitations when moving from one agent to two agents.

Chapter 9: Evaluation of Task-Adaptive Waypoint-Aware Transformer (TA-WAT) Policies in Dual-Agent Navigation

Building upon the architecture studies in the single-agent setting, this chapter examines the observed behaviour of the tested Transformer configurations in a dual-agent environment. In particular, it compares the Standard Transformer and Task-Adaptive Waypoint-Aware Transformer when moving from one agent to two agents, without claiming generalisation to larger multi-agent teams.

Chapter 10: Discussion, Contributions, and Conclusion

This chapter summarizes the key findings, discusses the limitations of the present work, and outlines potential directions for future research.

# Chapter 2

# Literature Review

Path planning presents a central challenge in robotics and autonomous navigation. It dictates how an autonomous agent or robot moves from the origin to the destination, ensuring both safety and efficiency while avoiding obstacles. Over recent decades, researchers have proposed a range of approaches, from classical search-based algorithms to modern deep reinforcement learning (DRL) methods.

## 2.1 Classical Path Planning Algorithms

Path planning is vital for autonomous navigation in complex environments, aiming to produce a feasible, optimal path from the origin to the destination in known or partially known environments. Traditional methods are prevalent in robotics, autonomous vehicles, and drone operations. This section examines the principles, evolution, and applications of these algorithms, highlighting strengths in static, known spaces and limitations in dynamic, uncertain environments.

### 2.1.1 Search, Planning and Mapping

This section examines search and mapping in path planning, connecting prior concepts to the critical role these mechanisms serve in algorithmic techniques. The search algorithm defines the exploration strategy and efficiency within an environment, while the mapping method determines feasibility and path optimality [26]. A feasible plan reaches the goal state, disregarding efficiency. An optimal plan achieves the goal and maximizes performance according to a given criterion [26]. Let's first examine basic search algorithms.

**Breadth-first search (BFS)** [27] inspects all neighbors at each level before progressing, guaranteeing a solution if one exists and optimality in unit-cost settings. Conversely, DFS pursues a single branch deeply before retreating, which can be more

memory-efficient but does not ensure the shortest path. Best-first search, in comparison, employs heuristics for more focused, goal-directed pursuit, though it may settle for local rather than global optima in large domains. Figure 2.1 illustrates the level-wise exploration pattern of BFS.

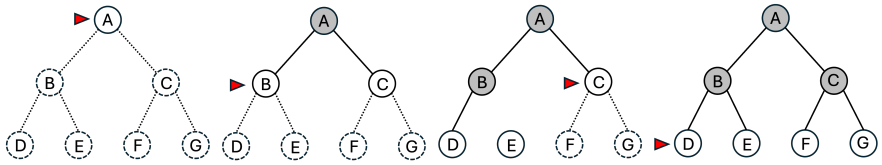



Figure 2.1: Breadth-first search on a simple binary tree.

**Depth-first search (DFS)** [27] fully explores one path before backtracking, recording explored nodes to avoid repetition. It returns the first, possibly suboptimal, solution found, as illustrated in Figure 2.2.

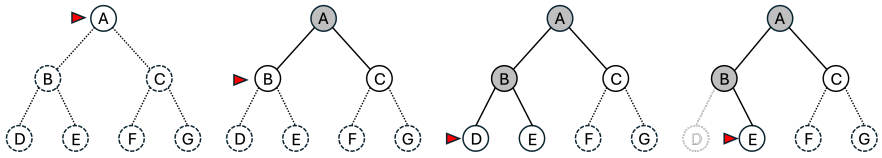



Figure 2.2: Depth-first search on a simple binary tree.

**Best-first search** [27] employs heuristics to prioritize the next node based on the lowest evaluation function value. Leveraging heuristic estimates—such as minimum distance—it steers the search more efficiently than uninformed strategies. While it boosts efficiency and avoids unproductive states, it may yield only a locally optimal outcome, and its computational requirements can grow rapidly in large search spaces.

### 2.1.2 Optimal Search

Optimal path-planning algorithms seek the shortest path while minimizing computational cost. Building on the foundational search techniques described earlier, this section introduces uninformed algorithms such as Dijkstra and Uniform Cost Search (UCS), leading to the creation of the A*algorithm, followed by advanced variants, including Iterative Deepening A* (IDA*) and bidirectional search. Let's begin with Dijkstra's algorithm.

**Dijkstra and Uniform Cost Search (UCS)** [28] are cost-oriented algorithms for computing the shortest path in weighted graphs. Both prioritize node expansion based on incremental path costs. Dijkstra's technique, introduced in 1959, assumes non-negative edge weights and consistently finds the optimal path by selecting the lowest-cost edge at each step. UCS [27] generalizes Dijkstra by handling arbitrary

costs through uniform expansion. Although both discover optimal paths, they rapidly increase computational and memory demands as graph complexity increases, limiting their applicability in high-dimensional, continuous spaces.

**A*algorithm** [18] merges Dijkstra's cost optimality with heuristic speed, using f(n) = g(n) + h(n). Unlike Dijkstra, which selects nodes solely on cost, A* exploits heuristics to focus the search, reducing expansions when the heuristic is well-calibrated. This typically makes A* faster and more scalable than uninformed algorithms but also more sensitive to heuristic accuracy and still constrained by high memory use in expansive settings.

**Iterative Deepening A* (IDA*)** [29] conserves memory compared to A* by employing repeated depth-limited searches with escalating thresholds. Although both A* and IDA* achieve optimal paths with admissible heuristics, IDA* revisits nodes multiple times, causing inefficiencies in dense graphs. Unlike BFS and Dijkstra, which require more memory, IDA* balances optimality with lower space demands but may incur higher computation in intricate networks.

**Bidirectional Search** [27] is an optimization method that accelerates search by simultaneously expanding the graph from both the start and goal nodes. Each search advances independently—one forward, one backward—until the two frontiers converge, forming a complete path. By effectively halving the search depth, this strategy can sharply reduce the number of nodes explored compared to unidirectional searches such as BFS or Dijkstra. However, it requires the problem space to support efficient predecessor creation and uniform path-cost evaluation in both directions. In practice, coordinating the two search frontiers can introduce additional computational overhead, especially in complex or high-dimensional settings.

As summarized in Table 2.1 , graph-based search algorithms are the cornerstone of traditional path planning, evolving from uninformed routines like BFS and DFS, which operate without extra information, to heuristic-driven methods such as A* and IDA*, and cost-based or bidirectional strategies, which leverage data to increase efficiency. All these algorithms guarantee completeness and optimality under specific conditions, yet uninformed methods often falter in large or complex domains. Heuristic and cost-based methods address some efficiency challenges but may still face scalability, memory, and adaptability constraints in dynamic or continuous environments. Recognizing these distinctions clarifies the rationale for exploring learning-based and adaptive methods in the later sections [18, 27, 26, 28].

### 2.1.3 Incremental and Anytime Search with Mapping

Incremental search algorithms use previous computations to rapidly update solutions when the environment shifts. Anytime search algorithms deliver an initial,

**Algorithm 1** A* Search [18]

```
1: procedure ASTAR(start, goal)
2:     open ← priority queue; insert start with priority f(start) = h(start)
3:     came_from ← {}                                  ▷ map: node → predecessor
4:     g[start] ← 0
5:     f[start] ← h(start)
6:     closed ← ∅
7:     while open is not empty do
8:         current ← node in open with lowest f
9:         if current = goal then
10:            return RECONSTRUCTPATH(came_from, current)
11:        end if
12:        remove current from open
13:        add current to closed
14:        for all neighbor in NEIGHBORS(current) do
15:            if neighbor in closed then
16:                continue
17:            end if
18:            tentative_g ← g[current] + DIST(current, neighbor)
19:            if neighbor not in open or tentative_g < g[neighbor] then
20:                came_from[neighbor] ← current
21:                g[neighbor] ← tentative_g
22:                f[neighbor] ← g[neighbor] + h(neighbor)
23:                if neighbor not in open then
24:                    insert neighbor into open with priority f[neighbor]
25:                end if
26:            end if
27:        end for
28:    end while
29:    return failure
30: end procedure
31: function RECONSTRUCTPATH(came_from, current)
32:    path ← [current]
33:    while current in came_from do
34:        current ← came_from[current]
35:        prepend current to path
36:    end while
37:    return path
38: end function
```

Table 2.1: Comparison of Classical Search Algorithms

| Algorithm | Type | Optimal | Complete |
|---|---|---|---|
| BFS | Uninformed | Yes (unit-cost) | Yes |
| DFS | Uninformed | No | No |
| Best-first Search | Informed | Depends on heuristic | No |
| UCS | Uninformed (cost-based) | Yes | Yes |
| A* | Informed (heuristic + cost) | Yes (admissible $h$) | Yes |
| IDA* | Informed (iterative deepening A*) | Yes (admissible $h$) | Yes |
| Bidirectional Search | Uninformed / Informed | Yes (with proper conditions) | Yes |

possibly suboptimal, solution swiftly and refine it as more computing time becomes available. Together, these approaches strive to preserve solution quality while reducing replanning effort in dynamic settings. This section starts from Lifelong Planning A* (LPA*) [20] and covers D* [30] and D* Lite [30] .

**Lifelong Planning A* (LPA*)** [20]extends the standard A*algorithm by recomputing only the section of environment that has changed, rather than recomputing the entire map. It maintains two key values for each node: g(n) (the current best path cost) and rhs(n) (the one-step lookahead cost), efficiently updating only the affected nodes when edge costs change. Reusing previous computations significantly reduces replanning time compared to A*. However, its performance degrades as the number of required changes increases, since each node still needs to be checked for consistency, which requires maintaining and updating a priority queue.

**D***(Dynamic A*) [30] builds on the concept of LPA* to support real-time path correction in environments with unknown or changing obstacles. Unlike A*, D* searches backwards from the goal and dynamically propagates cost changes through "RAISE" and "LOWER" operations when new obstacles are detected. This allows the agent to efficiently replan on the fly without restarting the entire search. Despite its real-time capability, D* suffers from algorithmic complexity and implementation difficulty due to its numerous update conditions and cost propagation rule.

**D* Lite** [30]simplifies D* by adopting the core ideas of LPA* while using the D* framework of searching backwards from the goal. It preserves the same ability to repair paths incrementally but replaces the intricate wave propagation with a more streamlined priority-based update mechanism. As a result, D* Lite achieves comparable performance with improved computational efficiency and simpler implementation. Nevertheless, dynamic environments that update frequently can still lead to excessive overhead in key reordering and node consistency checks.

In summary, these algorithms build upon the A*algorithm into incremental and anytime frameworks, capable of handling environmental changes and real-time constraints. Although they significantly improve computational efficiency through reusing

prior search efforts, they still face challenges in scalability, update overhead, and stability in rapidly changing environments [18, 30, 20].

### 2.1.4 Summary

A* was selected for the experiments because the environments used in this study are static, fully observable, and represented as discretized grid maps. Under these conditions, A* provides deterministic and reproducible shortest paths without requiring the incremental replanning mechanisms of D* or D* Lite. Its use also maintains methodological consistency with the reference A*-guided reinforcement-learning study [4]. A controlled comparison with alternative planners was outside the experimental scope and available computational resources of this study and is therefore left for future work.

As summarized in Table 2.2 , classical path-planning algorithms provide strong optimality guarantees in static and fully observable environments but lack adaptability and learning capability. While extensions improve replanning efficiency, these methods remain heuristic-driven and brittle under uncertainty. These limitations motivate the integration of planning-derived structure with learning-based approaches, which is further discussed in the context of deep reinforcement learning in the following sections.

Table 2.2: Comparison of Classical Pathfinding Algorithms

| **Algorithm** | **Optimal** | **Dynamic Environment** | **Computational Cost** |
|---|---|---|---|
| Dijkstra | Yes | No | High |
| A* | Yes | No | Medium |
| LPA* | Yes | Yes | Medium |
| D* Lite | Yes | Yes | Low |

## 2.2 Machine Learning

**Machine learning** is a subfield of artificial intelligence that focuses on the development of algorithms and statistical models enabling computers to improve their performance on a specific task through experience, without being explicitly programmed. As formally defined by Mitchell [31], "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." This definition highlights the core idea that learning involves measurable improvement through exposure to data. Recent overviews further emphasize the broad applicability and rapid evolution of machine learning techniques across domains [31].

As shown in Figure 2.3 , machine learning can be divided into three main types: supervised, unsupervised, and reinforcement learning.

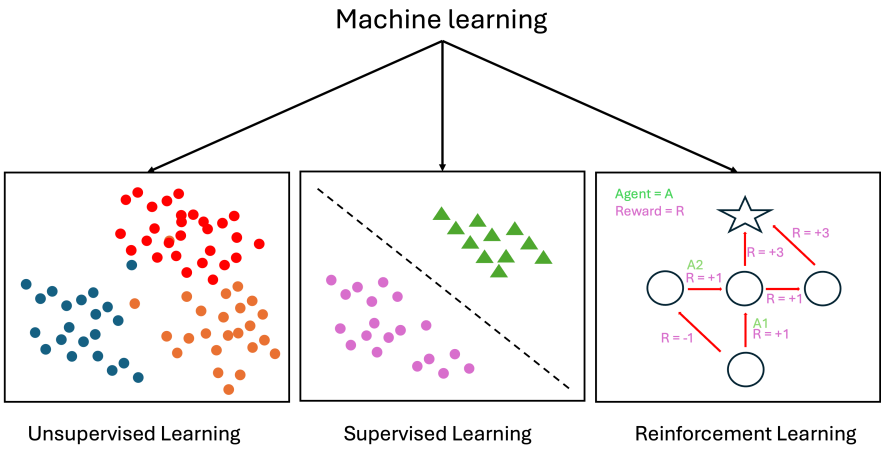



Figure 2.3: Machine learning classification.

**Unsupervised learning** uses unlabeled data to discover hidden patterns and groupings, such as clustering [31].

**Supervised learning** trains models using labelled data, where inputs are paired with known outputs. It is commonly used for classification and regression tasks [31].

**Reinforcement learning (RL)** is a learning paradigm where an agent interacts with an environment by taking actions and receives feedback in the form of rewards. The goal is to learn a strategy that maximizes cumulative rewards over time. Unlike supervised learning, RL learns from trial and error without explicit labeled examples [8]. This makes RL well-suited for sequential decision-making problems in robotics, games, and autonomous systems.

## 2.3 The Evolution of Neural Network Architectures

Neural networks are the core of deep learning, and there are various architectures that are good at handling different types of data and tasks. This chapter outlines the process from MLP to CNN and finally to the transformer.

### 2.3.1 Multilayer Perceptron(MLP)

Early neural network research was primarily based on multilayer perceptrons (MLPs), which consist of fully connected layers capable of approximating any continuous function via nonlinear activation functions. However, due to their dense connections, MLPs are computationally expensive and inefficient when processing structured data such as images or sequences. A representative MLP structure is shown in Figure 2.4.

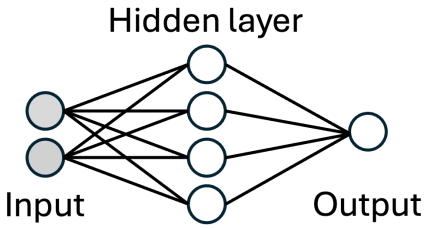



Figure 2.4: An MLP with one hidden layer.

### 2.3.2 Convolutional Neural Network(CNN)

To overcome these limitations, Convolutional Neural Networks (CNNs) were introduced, leveraging local receptive fields and weight sharing to dramatically reduce the number of parameters while preserving spatial correlations within the input data. By applying convolutional filters in a sliding-window manner, CNNs can extract hierarchical features—from edges and textures in early layers to more abstract patterns such as object parts and semantic structures in deeper layers [32]. This hierarchical representation makes CNNs highly effective for image classification, object detection, semantic segmentation, and various visual perception tasks. As a result, CNNs have become the foundational architecture in computer vision and remain widely used despite the recent rise of Transformer-based models. Figure 2.5 presents a representative CNN architecture.

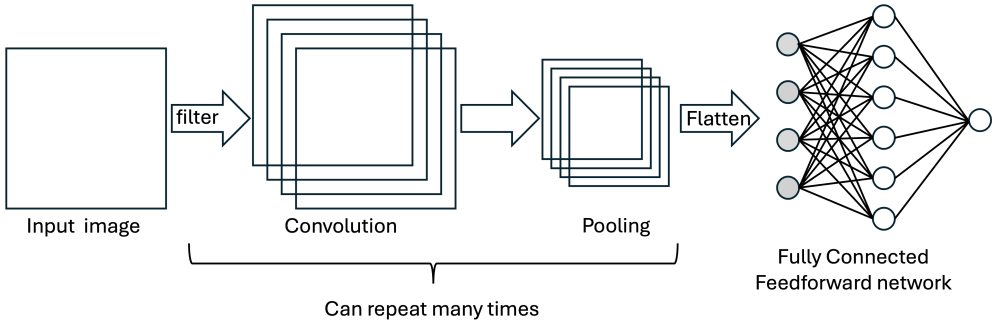



Figure 2.5: Architecture of a traditional CNN.

### 2.3.3 Transformer

While Convolutional Neural Networks (CNNs) effectively capture local patterns through shared kernels and hierarchical feature extraction, their reliance on limited receptive fields makes it difficult to model long-range dependencies without stacking many layers. This requirement increases computational depth and may be inefficient in tasks where global context is needed.

To address these limitations, the Transformer architecture replaces convolutional operations with self-attention mechanisms, enabling the model to directly compute global interactions between all elements in the input. This design eliminates the need for deep stacking to capture long-range dependencies and provides full parallelization during training. As a result, Transformers offer greater flexibility and scalability and have become a widely adopted alternative to convolution-based architectures for many sequence- and structured-data modeling tasks [1]. However, temporal dependency modeling requires observations from multiple timesteps to be explicitly included in the input. When a Transformer receives only tokens from the current observation, self-attention models relationships within that observation rather than temporal memory across previous states.

Transformer architectures can be categorized by how they process information through encoding and decoding pathways. Encoding-only models (e.g., BERT) focus on bidirectional feature extraction: they read the entire input simultaneously and learn contextualized representations by attending to tokens on both the left and right. This structure is well-suited for tasks requiring a deep understanding of an input sequence—such as classification, reasoning, or state representation learning in reinforcement learning—because the encoder builds rich latent embeddings that summarize the environment [33].

In contrast, decoding-only models (e.g., GPT-style architectures) generate outputs autoregressively: each token is produced based only on previously generated tokens. This design makes them ideal for sequence generation, planning, and long-horizon prediction. When applied in RL, decoder-only models can model trajectories, predict future actions, or generate temporally coherent plans by treating decision-making as a sequential generation problem. Together, encoding-only and decoding-only designs highlight two complementary roles of Transformers—understanding complex states and producing structured action sequences—providing flexible foundations for memory-augmented agents [33]. The general Transformer architecture is shown in Figure 2.6.

Although Transformer architectures have demonstrated remarkable success in sequence modeling and long-range dependency capture, their application in reinforcement learning remains non-trivial. Unlike supervised learning, RL involves non-stationary targets and sparse reward signals, which can destabilize attention-based models. This raises an open question as to whether Transformers can effectively replace MLPs in navigation-oriented DRL tasks without task-specific adaptations.

### 2.3.4 Summary

The evolution of neural network architectures reflects a shift from densely connected models to structures that better exploit input regularities and long-range dependencies.

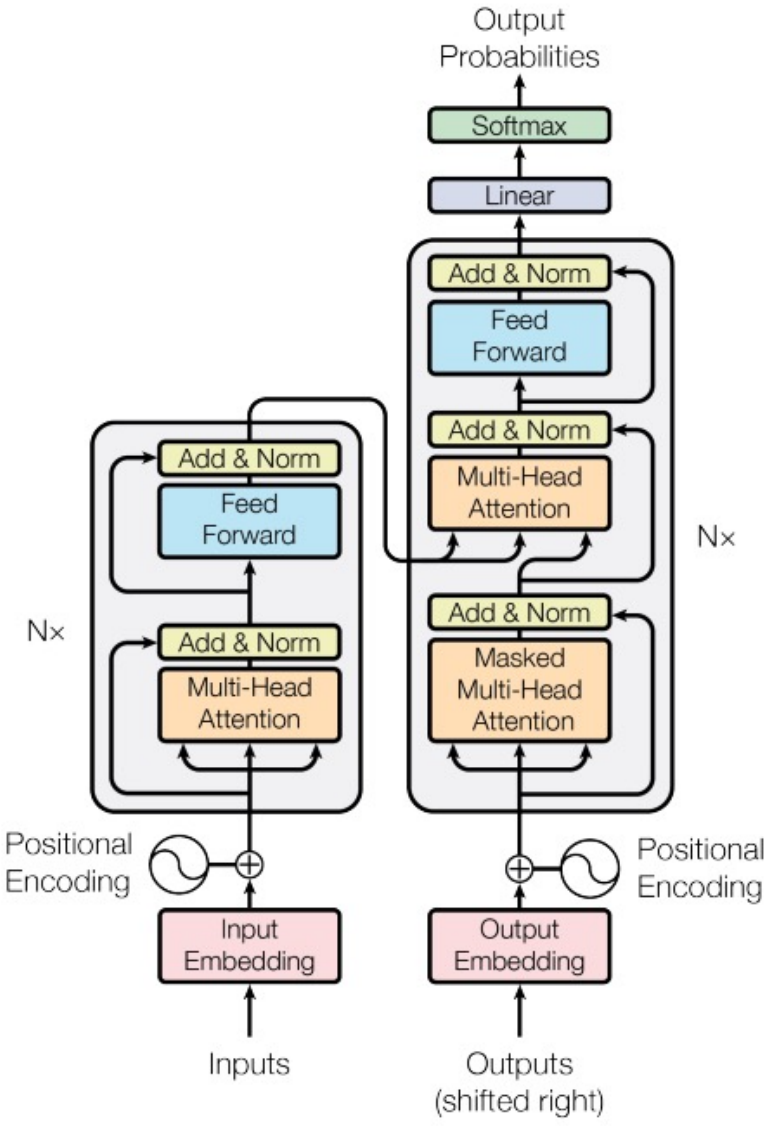



Figure 2.6: The Transformer - model architecture [1].

Early architectures, such as Multilayer Perceptrons (MLPs), provide universal function approximation but are inefficient for structured data due to dense parameterization and the lack of spatial locality. Convolutional Neural Networks (CNNs) address this limitation by introducing local receptive fields and weight sharing, allowing hierarchical feature extraction and making CNNs the dominant paradigm in computer vision for decades [32]. However, their limited receptive fields require deep stacking to model global relationships.

Transformers represent a major architectural breakthrough by replacing convolution with self-attention, enabling direct modeling of global dependencies and full parallelization during training [1]. Their encoder–decoder variants support both contextual understanding and autoregressive generation, making Transformers suitable for reasoning, planning, and reinforcement-learning tasks requiring long-horizon temporal modeling [1]. Together, these architectures illustrate a clear progression toward more expressive, scalable, and memory-capable models for navigation and sequential decision-making.

Table 2.3 compares the key mechanisms, strengths, and limitations of common neural network architectures. Table 2.4 further relates these architectural properties to their suitability for navigation and deep reinforcement learning tasks.

Table 2.3: Comparison of Neural Network Architectures

| **Architecture** | **Key Mechanism** | **Strengths** | **Limitations** |
|---|---|---|---|
| MLP | Fully connected layers | Universal approximator | Inefficient for structured data |
| CNN | Convolution + pooling | Locality, hierarchy, efficiency | Limited global context |
| Transformer | Self-attention | Global dependencies, parallelism | High computational cost |

Table 2.4: Suitability of NN Architectures for Navigation and DRL Tasks

| **Architecture** | **State Representation** | **Temporal Modeling** | **Use Cases** |
|---|---|---|---|
| MLP | Low-dimensional states | None | Grid-based, tabular features |
| CNN | Visual/spatial states | Weak | Image-based navigation, Atari |
| Transformer | High-dimensional + sequence | Conditional on temporal input | Long-horizon RL, planning |

## 2.4 Deep Reinforcement Learning

Deep reinforcement learning (DRL) refers to reinforcement learning methods that employ deep neural networks to approximate policies, value functions, or environment dynamics [2]. Figure 2.7 illustrates this interaction between deep networks and the reinforcement-learning loop.

### 2.4.1 Deep Reinforcement Learning Classification

Deep reinforcement learning (DRL) methods are commonly categorized into model-based and model-free approaches.

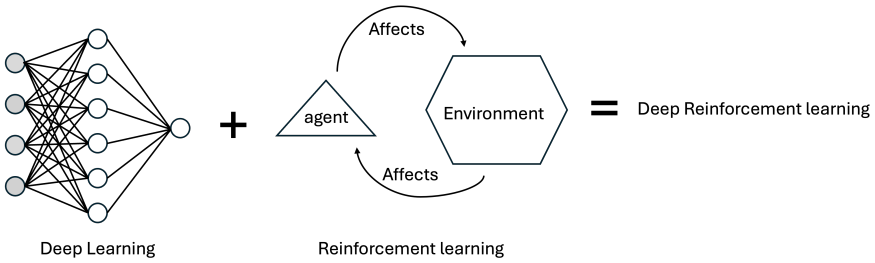



Figure 2.7: Deep Reinforcement Learning Diagram[2].

Model-based reinforcement learning leverages a learned or predefined dynamics model to generate effective policies and is therefore far more sample-efficient than model-free approaches. While it still requires real-world interaction to acquire or refine the environment model, its ability to reuse learned dynamics enables it to achieve competent behavior with significantly fewer samples. Nevertheless, model-based methods often suffer from poor asymptotic performance, as training focuses on approximating the environment dynamics rather than directly optimizing the policy for maximal long-term reward [34].

In contrast, model-free DRL, which directly learns a policy or value function from interactions without constructing an explicit model, has become the dominant paradigm for many complex control and multi-agent scenarios due to its stability and ease of implementation. The present study focuses exclusively on model-free methods [2].

Upon this foundation, it is necessary to further distinguish how different model-free approaches represent and optimize decision policies. Deep reinforcement learning (DRL) algorithms can be broadly categorized according to how they represent and optimize policies. In the existing literature, DRL methods are commonly grouped into three major families: value-based, policy-based, and actor–critic approaches. Figure 2.8 summarizes this taxonomy, and the following subsections present the methodological differences and development of the methods within each family [3].

### 2.4.2 Value-Based Methods: From Q-Learning to Rainbow DQN

**Q-learning** [2] is a value-based reinforcement learning method. The neural network estimated the expected return (value) for each possible action in a given state, compared these estimates with the observed rewards, and adjusted its parameters accordingly. This procedure represents a specific and simplified instance of the general family of Q-learning methods, whose update rule is illustrated in Figure 2.9.

**The Deep Q-Network(DQN)** [35] established a practical framework for combining Q-learning with deep convolutional models and demonstrated that agents can learn directly from high-dimensional visual input. By introducing experience replay and

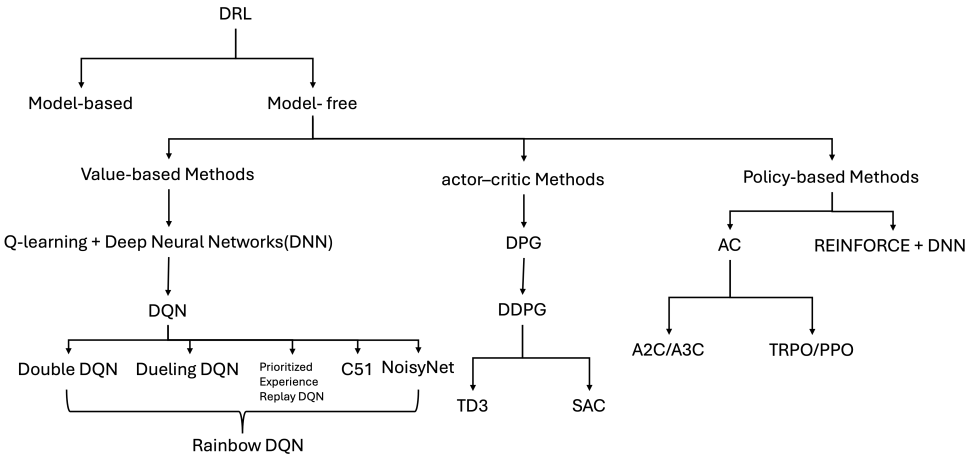



Figure 2.8: Taxonomy of major model-free deep reinforcement learning methods [3].

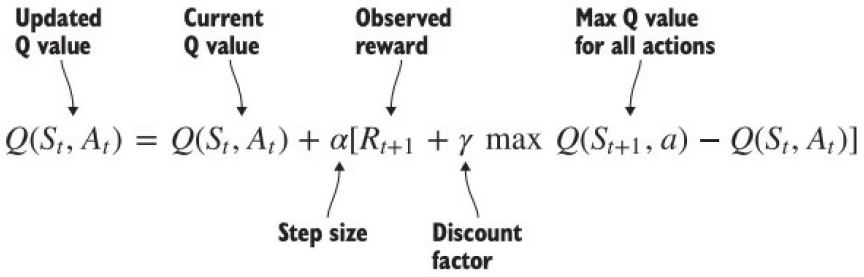



Figure 2.9: Q-learning update rule [2].

**Algorithm 2** Q-value Update Rule [2].

**Require:** Current Q-value $Q(s_t, a_t)$, reward $r_{t+1}$, next state $s_{t+1}$, step size $\alpha$, discount factor $\gamma$

1: $target \leftarrow r_{t+1} + \gamma \max_a Q(s_{t+1}, a)$
2: $td\_error \leftarrow target - Q(s_t, a_t)$
3: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot td\_error$

a target network, DQN alleviated the instability issues that had previously hindered value-based deep reinforcement learning. These mechanisms reduce update correlation and slow the drift of bootstrap targets, both of which contribute to more reliable training dynamics. As a result, DQN achieved strong human-level performance across many Atari games and served as the foundation for subsequent improvements.

**Algorithm 3** Deep Q-Network (DQN) [36, 6, 35]

Initialize replay buffer $\mathcal{D}$ and Q-network $Q(\cdot;\theta)$
Initialize target network $\hat{Q}(\cdot;\theta^-)$ with $\theta^- \leftarrow \theta$
**for** each episode **do**
    Observe initial state $\phi_1$
    **for** $t = 1$ to $T$ **do**
        Select action $a_t$ using $\varepsilon$-greedy policy from $Q(\phi_t, \cdot; \theta)$
        Execute $a_t$, observe $r_t$ and $\phi_{t+1}$
        Store $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample minibatch $(\phi_j, a_j, r_j, \phi_{j+1}) \sim \mathcal{D}$
        $y_j \leftarrow r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-)$
        Update $\theta$ by minimizing

$$L = (y_j - Q(\phi_j, a_j; \theta))^2$$

        Periodically update target network: $\theta^- \leftarrow \theta$
    **end for**
**end for**

While effective, the original DQN suffers from significant overestimation because it uses a single network to both select and evaluate actions. **Double DQN** [35]resolves this issue by decoupling these two processes: the online network selects the greedy action, and the target network evaluates it. This modification substantially reduces overoptimistic value estimates that can destabilize learning. Empirically, Double DQN produces more stable Q-functions and improves performance in many Atari environments.

Uniform sampling from replay memory treats all transitions equally, limiting data efficiency. **Prioritized Experience Replay** [35] instead assigns higher sampling probability to transitions with larger temporal-difference errors, allowing the agent to focus updates on samples that provide stronger learning signals. Importance sampling weights are used to correct the bias introduced by this non-uniform selection. This approach significantly speeds up training and often yields better final policy quality.

**The dueling architecture** [35]separates the estimation of the state value from

the advantage of each action, allowing the model to learn which states are important even when actions have similar outcomes. This decomposition encourages better representation learning, especially in environments with redundant or irrelevant actions. By aggregating the value and advantage streams into Q-values using a stable formulation, it improves generalization across actions. Experiments show that the dueling structure enhances learning efficiency and boosts overall performance.

**Multi-step learning** [35]replaces the conventional one-step TD target with an n-step return, enabling reward information to propagate more quickly through time. This richer return structure offers a beneficial trade-off between bias and variance, particularly with moderate values of n. By accelerating the transmission of value signals, multi-step updates tend to improve both early learning speed and ultimate policy quality. In practice, combining multi-step returns with deep Q-learning significantly enhances the agent's effectiveness.

**Distributional RL** [35]extends value-based learning by predicting the entire distribution of future returns rather than only their expectation. Modeling value as a categorical probability distribution allows the agent to capture richer statistical information about reward outcomes. This leads to more informative learning targets and more expressive value functions. Empirical evidence shows that distributional representations substantially increase performance in many Atari games.

**Noisy Networks** [35] address the limitations of $\varepsilon$-greedy exploration by injecting trainable, parameterized noise directly into network weights. This approach enables exploration that adapts to the current state and evolves throughout training, allowing the agent to balance exploration and exploitation more effectively. As the noise parameters are learned, the agent can automatically reduce exploratory behavior when no longer needed. This results in more directed exploration and improved performance in challenging environments.

**Rainbow** [35] unifies all the above components into a single architecture, showing that these techniques interact in largely complementary ways. The agent leverages multi-step learning within a distributional framework, applies double Q-learning to distributional updates, and uses prioritized replay with KL-based priorities. Noisy layers replace $\varepsilon$-greedy, and a dueling structure strengthens state representation learning. Combined, these components enable Rainbow to achieve substantial gains in data efficiency and state-of-the-art performance across the full Atari benchmark.

### 2.4.3 Policy-Based Methods: From REINFORCE to PPO

**REINFORCE** [8] is a Monte Carlo policy gradient method that updates policy parameters by correlating sampled returns with the log-likelihood of taken actions. It

provides an unbiased estimator of the true policy gradient without requiring value function approximation. However, because returns can be highly variable across episodes, REINFORCE typically suffers from high variance and slow convergence. Despite its simplicity, REINFORCE laid the foundation for modern policy gradient methods and remains a core example of gradient-based stochastic control.

**Algorithm 4** REINFORCE Algorithm [8]

Initialize policy parameters $\theta$
**for** each episode **do**
    Generate a trajectory $\tau = (s_0, a_0, r_1, \ldots, s_T)$ using $\pi_\theta$
    **for** each step $t$ in episode **do**
        Compute return $G_t = \sum_{k=t}^{T} \gamma^{k-t} r_{k+1}$
        Update policy:

$$\theta \leftarrow \theta + \alpha \, G_t \nabla_\theta \log \pi_\theta(a_t|s_t)$$

    **end for**
**end for**

**REINFORCE with baseline** [8] improves the original algorithm by introducing a learned baseline, typically a state-value estimate, to reduce the variance of policy updates. The baseline does not affect the expected gradient, ensuring the estimator remains unbiased, but it significantly stabilizes training and accelerates convergence. This variance-reduction mechanism forms the foundation of modern actor–critic methods, in which the critic serves as a learned baseline to guide policy updates more effectively.

**Algorithm 5** REINFORCE with Baseline [8]

Initialize policy parameters $\theta$ and baseline parameters $\phi$
**for** each episode **do**
    Generate a trajectory $\tau$ using $\pi_\theta$
    **for** each step $t$ **do**
        Compute return $G_t$
        Estimate baseline $b_t = V_\phi(s_t)$
        Update policy:

$$\theta \leftarrow \theta + \alpha(G_t - b_t)\nabla_\theta \log \pi_\theta(a_t|s_t)$$

        Update baseline by regression toward $G_t$:

$$\phi \leftarrow \phi - \beta\nabla_\phi(G_t - V_\phi(s_t))^2$$

    **end for**
**end for**

**Actor–critic algorithms(AC)** [8]combine the strengths of policy-based and value-based reinforcement learning by maintaining two complementary components: an actor that directly parameterizes the policy and a critic that estimates value functions such as $V(s)$ or $Q(s, a)$. This structure enables low-variance, more stable policy gradient estimation, since the critic serves as a learned baseline that guides the actor toward beneficial updates. Unlike REINFORCE, which relies solely on Monte-Carlo returns, actor–critic methods incorporate bootstrapping, improving data efficiency and reducing the variance–bias trade-off. The framework generalizes naturally into modern algorithms such as A2C [37], A3C [37], DDPG [38], TD3 [39], and PPO [37], forming one of the most influential foundations in deep reinforcement learning.

**A3C (Asynchronous Advantage Actor-Critic)** [37] extends the actor–critic framework by running multiple asynchronous worker agents, each interacting with its own environment instance and independently updating a shared global network. This design decorrelates training data and mitigates stability issues seen in single-threaded actor–critic methods. **A2C (Advantage Actor-Critic)** [37], the synchronous variant of A3C, aggregates gradients from multiple parallel workers and applies them in a single synchronized update, removing the gradient-race conditions of A3C while preserving its throughput and variance-reduction benefits. This design decorrelates training data, mitigates stability issues seen in single-threaded actor–critic methods, and leverages multi-step bootstrapped returns to accelerate value propagation, as highlighted in the literature where A3C is used to illustrate effective $n$-step learning strategies. **A2C** [37],

the synchronous variant of A3C, aggregates gradients from multiple parallel workers and applies them in a single synchronized update, removing the gradient-race conditions of A3C while preserving its throughput and variance-reduction benefits. Together, A2C/A3C represent a practical and efficient family of actor–critic algorithms widely used in environments with unlimited states and discrete actions, and are reported as competitive baselines in contemporary deep RL benchmarks, including Atari.

**Trust Region Policy Optimization (TRPO)** [37]addresses the instability of vanilla policy gradient updates by enforcing a constraint on how much the policy is allowed to change at each iteration. TRPO optimizes a surrogate objective while restricting the KL-divergence between the new and old policies, ensuring monotonic policy improvement and preventing large, destructive updates that could collapse performance. Although this trust-region formulation provides strong theoretical guarantees, it requires solving a constrained optimization problem using second-order methods, which increases computational overhead and limits scalability in deep RL settings.

**Proximal Policy Optimization (PPO)** [37]simplifies TRPO's trust-region idea into a more practical and computationally efficient algorithm by introducing either a KL-penalty or the widely adopted clipped surrogate objective. The clipping mechanism limits the policy ratio's deviation, effectively approximating TRPO's constraint with only first-order optimization, enabling fast minibatch SGD and improving ease of implementation. PPO maintains stable learning, reduces destructive policy updates, and achieves strong empirical performance across continuous and discrete control domains, making it one of the most widely used on-policy algorithms in modern deep reinforcement learning. Algorithm 6 summarizes the standard PPO training procedure. Implementation details such as learning rate schedules and clipping variants follow common practice and are omitted for brevity.

**Algorithm 6** Proximal Policy Optimization (PPO) [37, 40]

**Require:** policy parameters $\theta$, value parameters $\phi$ (possibly shared backbone)

**Require:** clip factor $\varepsilon$, value loss coeff $c_1$, entropy coeff $c_2$

**Require:** discount $\gamma$, GAE parameter $\lambda$ (optional), epochs $K$, minibatch size $M$

Initialize $\theta, \phi$

**while** not converged **do**

  Collect a set of trajectories $\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1})\}$ by running policy $\pi_{\theta_{\text{old}}}$

  Compute returns $R_t = \sum_{l=0}^{T-t-1} \gamma^l r_{t+l}$ (or bootstrapped $n$-step returns)

  Compute advantage estimates $\hat{A}_t$ (e.g. GAE):

$$\delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t), \qquad \hat{A}_t = \sum_{l=0}^{T-t-1} (\gamma\lambda)^l \delta_{t+l}$$

  Normalize advantages: $\hat{A}_t \leftarrow (\hat{A}_t - \text{mean})/(\text{std} + 10^{-8})$

  **for** $k = 1 \dots K$ (epochs) **do**

    Shuffle $\mathcal{D}$ and partition into minibatches of size $M$

    **for** each minibatch $B$ from $\mathcal{D}$ **do**

      Compute probability ratios:

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

      Clipped surrogate objective:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_{t\in B}\Big[\min\big(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t\big)\Big]$$

      Value function loss:

$$L^{\text{VF}}(\phi) = \mathbb{E}_{t\in B}\big[(V_\phi(s_t) - R_t)^2\big]$$

      Entropy bonus:

$$S[\pi_\theta](s_t) = -\mathbb{E}_{a\sim\pi_\theta(\cdot|s_t)}\big[\log \pi_\theta(a|s_t)\big]$$

      Total (to *maximize*):

$$L(\theta, \phi) = L^{\text{CLIP}}(\theta) - c_1 L^{\text{VF}}(\phi) + c_2 \mathbb{E}_{t\in B}[S[\pi_\theta](s_t)]$$

      Perform gradient ascent (or descent on negative loss) on $\theta, \phi$ using minibatch gradients

    **end for**

  **end for**

  $\theta_{\text{old}} \leftarrow \theta$

**end while**

### 2.4.4 Actor–Critic Methods: From DPG to SAC

**Deterministic Policy Gradient (DPG)** [41] introduces a deterministic alternative to stochastic policy gradient methods by learning a policy that maps states directly to specific actions rather than to action distributions. This formulation greatly reduces variance in gradient estimates and enables efficient learning in environments with continuous action spaces. DPG optimizes the deterministic policy by backpropagating through a learned action-value function, computing gradients of $Q(s, a)$ with respect to actions, and then propagating them through the policy network. Combined with off-policy sampling, DPG allows the use of replay buffers and target networks for stability, forming the foundation for the widely used Deep Deterministic Policy Gradient (DDPG) algorithm. The deterministic framework improves sample efficiency and scalability in high-dimensional continuous control tasks, making DPG a cornerstone of modern actor–critic algorithms for continuous action reinforcement learning.

**Deep Deterministic Policy Gradient (DDPG)** [38] is an off-policy deterministic actor–critic algorithm designed for continuous action spaces. It combines a deterministic policy with a target critic and replay buffer, enabling stable training via temporal-difference learning. However, as highlighted in subsequent work such as TD3 [39], DDPG suffers from significant overestimation bias due to function approximation and the use of a single critic, leading to unstable learning and poor performance across many tasks. Its vulnerability to critic error amplification and sensitivity to hyperparameters motivated later improvements, including double critics, delayed updates, and target smoothing.

**Twin Delayed Deep Deterministic Policy Gradient (TD3)** [39] addresses the function approximation errors and overestimation bias that commonly arise in deterministic actor–critic algorithms such as DDPG. The authors show that value overestimation also exists in continuous control settings and propose three key mechanisms: clipped Double Q-learning with two critics to reduce overestimation, delayed policy updates to allow the critic to stabilize before updating the actor, and target policy smoothing by adding clipped noise to the target action. These components collectively mitigate bias and variance accumulation in temporal-difference updates. Empirically, TD3 achieves state-of-the-art performance across a suite of continuous control tasks in OpenAI Gym, significantly outperforming previous actor–critic methods.

**Soft Actor-Critic (SAC)** [42] is an off-policy actor–critic algorithm developed under the maximum entropy reinforcement learning framework, which augments the traditional return objective with an entropy term to promote stochastic and exploratory behavior. By jointly maximizing expected reward and policy entropy, SAC promotes robustness and prevents premature convergence to suboptimal deterministic actions.

Its formulation integrates three core components: a stochastic actor that outputs reparameterized continuous actions, a pair of Q-functions that reduce overestimation bias, and a soft value function that provides a stable training target. Through soft policy iteration, SAC alternates between policy evaluation and improvement using off-policy samples, enabling efficient reuse of past experience. This design yields both high sample efficiency and strong stability, allowing SAC to outperform prior on-policy and off-policy algorithms on complex continuous-control benchmarks. Empirical evidence demonstrates that entropy maximization not only accelerates exploration but also enhances final performance across diverse environments.

### 2.4.5 Summary

Reinforcement Learning provides a general framework for agents to learn from trial-and-error interaction. Value-based and policy-based methods capture different optimization perspectives: the former focuses on estimating state or action values, while the latter directly optimizes a parameterized policy. Although RL is theoretically grounded, its performance is highly sensitive to reward sparsity, exploration difficulty, and unstable gradients. These limitations become more pronounced in navigation tasks that involve complex spatial constraints. Table 2.5 summarizes the principal method categories.

Table 2.5: Categories of Reinforcement Learning Methods

| **Category** | **Description** | **Examples** |
|---|---|---|
| Value-based | Learns value function | Q-learning, DQN |
| Policy-based | Directly optimizes policy | REINFORCE, PPO |
| Actor–Critic | Combines value + policy | A2C, SAC, TD3 |

Deep RL leverages neural networks to approximate value functions or policies. DQN stabilizes value-based learning using target networks and replay buffers [35], while PPO improves policy optimization stability with its clipped objective [37]. Actor–critic methods such as DDPG, TD3, and SAC extend DRL to continuous control [8, 38, 39, 42]. Table 2.6 compares representative DRL algorithms by action space, stability, and sample efficiency. Nonetheless, DRL methods remain sensitive to hyperparameters and require substantial interaction data, especially in sparse-reward navigation tasks.

While a wide range of DRL algorithms have been proposed, their effectiveness in navigation tasks is highly sensitive to reward design, exploration strategy, and policy representation. This variability complicates fair comparison and motivates controlled experimental evaluation under a unified framework.

Table 2.6: Comparison of Representative DRL Algorithms

| Algorithm | Action Space | Stability | Sample Efficiency |
|---|---|---|---|
| DQN | Discrete | Medium | Low |
| PPO | Discrete/Continuous | High | Medium |
| DDPG | Continuous | Low | Medium |
| TD3 | Continuous | High | Medium |
| SAC | Continuous | High | High |

## 2.5 Deep Reinforcement Learning in pathfinding

### 2.5.1 Platform Selection and Environment Setup

Selecting an appropriate platform is the foundation of a reproducible and efficient DRL workflow. *Deep Reinforcement Learning in Action* highlights the advantages of using standard toolchains such as Python, PyTorch, and OpenAI Gym, particularly their unified interfaces and broad community support [2]. These platforms provide consistent APIs (`reset()`, `step()`, `render()`) that simplify experimentation while enabling rapid prototyping and debugging.

In practice, ensuring deterministic behavior through fixed random seeds, logging utilities, and version control is essential for reproducibility. Researchers should also verify that the environment exposes all necessary state information and runs at sufficient simulation speed to support large-scale DRL training. Attention should be paid to software versions and hardware differences, which may otherwise introduce uncontrolled variability into results.

### 2.5.2 Problem Modeling: States, Actions, and Termination

Modeling a task as a Markov Decision Process (MDP) is a critical early step. The book emphasizes clearly defining the state and action spaces, the reward structure, and the episode termination criteria before selecting an algorithmic solution [2]. Properly designed states must capture all information required for effective decision-making, while action definitions should reflect the agent's actual control capabilities, whether discrete or continuous.

Misaligned termination conditions or poorly structured state representations can severely degrade training stability. When tasks exhibit sparse rewards, additional mechanisms such as $n$-step returns or intrinsic rewards may be introduced—but only after establishing a minimal, verifiable MDP formulation. The guiding principle is to begin with the simplest correct model and extend it only when necessary [2].

### 2.5.3 Network Architecture Selection and Comparison

Network architecture should be chosen based on the structure of observations and the complexity of the task. For low-dimensional, tabular-like inputs, simple multilayer perceptrons are sufficient. For visual observations such as Atari frames, convolutional neural networks provide strong representation learning capability. The book further illustrates how actor–critic architectures integrate policy and value estimation to improve sample efficiency and stability [2].

More advanced models—such as attention or relational networks—are appropriate only when the task inherently requires modeling interactions among entities. Practitioners should prioritize minimal architectures with reliable training behavior before exploring more complex designs. It is also important to ensure that input normalization, architecture capacity, and hardware constraints are aligned with the task scale [2].

### 2.5.4 Algorithm Configuration and Hyperparameter Tuning

Algorithm configuration is one of the most influential factors in DRL performance. *Deep Reinforcement Learning in Action* repeatedly notes the sensitivity of DRL methods to hyperparameters such as learning rate, exploration parameters, discount factor, batch size, replay buffer size, and $n$-step return length [2]. A structured tuning process—modifying only the most impactful parameters first—is crucial for avoiding confounded results.

Reproducibility requires consistent random seeds, careful version tracking, and sufficient logging. Stability techniques such as target networks or experience replay must be correctly applied when using DQN-based methods. Ultimately, the goal is to achieve stable learning curves first, then refine performance and sample efficiency through incremental tuning [2].

### 2.5.5 Reward Function Design and Optimization

Reward design directly shapes agent behavior and is often more important than architectural choices. The book emphasizes rewarding the true task objective rather than intermediate metrics and introducing penalties only when they discourage undesirable behavior [2]. Sparse-reward environments may require supplementary mechanisms, such as intrinsic motivation or prediction-error-based rewards, but these should be introduced cautiously.

During implementation, researchers must check reward scales, consistency, and whether the rewards inadvertently bias the agent toward unintended behavior. Reward shaping should be minimal and interpretable; overly complex reward structures

risk leading the agent to optimize against the wrong objective [2].

### 2.5.6 Evaluation Metrics and Benchmarks

Proper evaluation must reflect policy quality, stability, and generalization. Standard metrics include episode return, success rate, convergence speed, variance across seeds, and sample efficiency. Benchmarks such as CartPole, MountainCar, and Atari provide reproducible baselines widely used in DRL research [2].

A rigorous evaluation protocol requires running multiple seeds, separating training and testing environments, and reporting confidence intervals when possible. Raw returns across different tasks should not be compared directly unless reward scales are normalized. Clear documentation of evaluation settings is essential for reproducibility and fair comparison [2].

### 2.5.7 Summary

Single-agent DRL achieves competitive navigation performance in both grid-based and continuous spaces. However, challenges such as sparse rewards, dead-ends, and insufficient exploration limit its effectiveness in complex maze-like environments. Classical planners can provide structured guidance or reward shaping to alleviate these issues, forming the basis for hybrid planning-learning approaches. Table 2.7 summarizes the corresponding single-agent DRL navigation pipeline.

Table 2.7: Single-Agent DRL Navigation Pipeline

| **Component** | **Description** |
|---|---|
| Environment | Provides states, transitions, and rewards |
| Observation | Agent receives partial or full state |
| Policy Network | Produces action distribution or values |
| Action Selection | Agent selects action based on policy |
| Learning Update | Policy/value updated via RL algorithm |

## 2.6 Multi-Agent Pathfinding

### 2.6.1 Challenges in Cooperative Multi-Agent Reinforcement Learning

In cooperative multi-agent tasks such as Multi-Agent Path Finding (MAPF), multiple agents learn concurrently within a shared environment, causing their policies to evolve over time. From the perspective of any individual agent, this results in a non-stationary environment, violating the stationarity assumption required by standard reinforcement learning algorithms. Prior work has shown that independent learning

approaches, such as Independent Q-Learning (IQL) [43]., often misinterpret the exploration behavior of other agents as environmental stochasticity, leading to unstable training dynamics and poor coordination performance, even in relatively simple cooperative tasks [13, 15].

A natural solution to mitigate non-stationarity is centralized training, where learning is performed using global information such as the joint state and joint actions of all agents. By leveraging this additional information during training, centralized learning provides a more stable and informative optimization signal, particularly in cooperative settings with shared rewards. However, fully centralized approaches suffer from severe scalability limitations. As the number of agents increases, the joint action space grows exponentially, making learning computationally intractable and often resulting in suboptimal behaviors, such as the lazy agent phenomenon, where some agents contribute minimally to the overall task [13, 14].

### 2.6.2 Centralized Training with Decentralized Execution (CTDE)

To balance the stability benefits of centralized training with the scalability requirements of decentralized systems, many modern multi-agent reinforcement learning (MARL) algorithms adopt the Centralized Training with Decentralized Execution (CTDE) paradigm. Under CTDE, agents are trained using centralized information, including the global state, joint actions, and shared rewards, while execution remains fully decentralized [15].

During training, centralized critics or joint value functions condition on global information to alleviate non-stationarity and improve credit assignment among agents. In cooperative tasks with shared rewards, this centralized perspective enables the learning algorithm to more accurately evaluate the contribution of each agent's actions to the overall outcome, thereby facilitating more effective coordination and stable convergence [44].

During execution, however, each agent selects actions based solely on its local observations, without access to global information or other agents' policies. This decentralized execution mechanism ensures scalability and enables practical deployment in large-scale and partially observable environments. Representative CTDE-based methods include Value Decomposition Networks (VDN) [13], QMIX [14], Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [15], and Multi-Agent Proximal Policy Optimization (MAPPO) [16], all of which have demonstrated strong performance in cooperative multi-agent settings.

### 2.6.3 Independent Q-Learning (IQL)

**Independent Q-Learning (IQL)** is one of the earliest and simplest baselines in multi-agent reinforcement learning. Each agent treats other agents as part of the environment and applies standard single-agent Q-learning independently. However, because all agents update their policies simultaneously, the environment becomes non-stationary, violating the Markov property required for Q-learning convergence [43]. Claus and Boutilier (1998) provide a foundational analysis showing that independent learners often fail to distinguish environmental stochasticity from other agents' policy changes, leading to unstable learning dynamics and convergence to suboptimal equilibria in coordination tasks. These limitations highlight why more advanced frameworks—especially CTDE—became essential for achieving stable learning in cooperative MARL.

### 2.6.4 Value Decomposition Networks (VDN)

**Value-Decomposition Networks (VDN)** [13]propose a simple yet effective factorization of the team action-value function into a sum of per-agent value functions. By training agents with a joint team reward and backpropagating the global TD error into each agent's local value estimate, VDN enables centralized training without requiring individualized reward signals. This additive decomposition mitigates non-stationarity and credit assignment challenges encountered by independent learners, while maintaining fully decentralized execution at test time. Although less expressive than nonlinear factorizations like QMIX, VDN offers a lightweight architecture that reliably promotes cooperative behavior across agents.

### 2.6.5 QMIX

**QMIX** [14] enhances value-decomposition methods by representing the joint action-value function through a monotonic mixing network that combines individual agent utilities. The mixing network enforces a structural constraint ensuring that the joint value increases monotonically with each agent's local Q-value, guaranteeing that decentralized greedy action selection is aligned with the centralized critic. By conditioning the mixing weights on global state information via hypernetworks, QMIX captures complex inter-agent interactions while remaining scalable to larger teams. This monotonic value factorization enables rich centralized value functions to guide decentralized execution, significantly improving coordination in partially observable cooperative tasks. QMIX employs a monotonic value decomposition to enable centralized training with decentralized execution [14]. Implementation details are provided in Appendix A.1.1.

### 2.6.6 MADDPG

**MADDPG** [15] extends deterministic actor–critic methods to multi-agent systems by introducing a centralized critic for each agent while keeping decentralized actors for execution. During training, each critic conditions on the joint observations and actions of all agents, effectively stabilizing learning in the presence of non-stationarity induced by co-learning teammates or opponents. This centralized-training–decentralized-execution (CTDE) scheme enables effective credit assignment in mixed cooperative–competitive environments and allows actors to rely solely on local observations at test time. By leveraging full state and action information only during training, MADDPG mitigates non-stationarity and variance issues inherent to independent learners, yielding more robust coordination strategies in continuous multi-agent domains.

### 2.6.7 MAPPO

**MAPPO** [16] adapts Proximal Policy Optimization to cooperative multi-agent environments under the CTDE paradigm. It employs a centralized value function that conditions on global state information, reducing variance and improving the stability of on-policy updates, while the decentralized policies rely only on local observations. Despite PPO being traditionally considered less sample-efficient, MAPPO demonstrates unexpectedly strong performance in complex multi-agent benchmarks by combining parameter sharing, advantage normalization, and appropriately tuned clipping. This design preserves the theoretical benefits of PPO—such as controlled policy divergence—while enabling agents to learn coordinated behaviors efficiently through shared global value estimation. Its recurrent variant incorporates recurrent policies to address partial observability at increased training cost in Appendix A.1.2.

### 2.6.8 Summary

Multi-agent reinforcement learning introduces non-stationarity, credit assignment difficulties, and coordination challenges. The CTDE framework addresses these issues by enabling agents to share global information during training while maintaining independent execution. Algorithms such as IQL, QMIX, MADDPG, and MAPPO leverage CTDE in different ways, yet still face difficulties in complex navigation tasks with sparse rewards and dynamic interactions [14, 15, 16]. Table 2.8 summarizes their use of centralized training and decentralized execution.

Table 2.8: Overview of MARL Algorithms under CTDE Framework

| Algorithm | Type | Centralized Training | Decentralized Execution |
|---|---|---|---|
| IQL | Independent | No | Yes |
| VDN | Value Decomposition | Yes | Yes |
| QMIX | Value Decomposition | Yes | Yes |
| MADDPG | Actor–critic | Yes | Yes |
| MAPPO | Actor–critic | Yes | Yes |

## 2.7 Latest Relevant Works

This section reviews six recent studies (2021–2025) that integrate classical path planning algorithms, particularly A*, with deep reinforcement learning approaches for robot navigation and multi-agent coordination.

### 2.7.1 PPO-Based Dynamic Path Planning with A* Integration

Jin et al. [45] proposed a dynamic path planning algorithm that combines Proximal Policy Optimization (PPO) with A* pathfinding for mobile robots.

**Methodology:** The approach employs a two-stage architecture where A* first generates a globally optimal reference path in known static environments, and then the PPO agent learns to dynamically adjust the path in response to moving obstacles. The state representation includes the robot's current position, velocity, and local sensor readings, while the action space consists of velocity commands. The reward function incorporates path-following accuracy, collision avoidance, and goal-reaching objectives.

**Experimental Results:** Experiments conducted on simulated mobile robot platforms demonstrated that the hybrid approach achieves 15–20% higher success rates compared to pure A* planning in environments with dynamic obstacles. The method also showed faster adaptation to sudden environmental changes.

**Limitations:** The study is limited to single-agent scenarios with relatively simple obstacle dynamics. The computational overhead of running A* at each planning cycle is not thoroughly analyzed.

**Difference from Proposed Algorithm:** Unlike Jin et al.'s sequential A*-then-PPO approach, our proposed method integrates A*-generated distance information directly into the intrinsic reward signal, enabling end-to-end learning. Furthermore, we extend the framework to multi-agent settings with CTDE paradigms (VDN, QMIX, MAPPO), whereas Jin et al. focus exclusively on single-agent PPO.

### 2.7.2 Hybrid MCTS–A* Path Planning

Zhang [46] proposed a hybrid obstacle-avoidance path planning method for intelligent mobile robots operating in environments with dense static obstacles. The approach

integrates Monte Carlo Tree Search (MCTS) with the classical A*algorithm to improve planning efficiency and path quality in cluttered grid-based environments.

**Methodology:** The method adopts a hierarchical planning framework in which A* is first used to generate a global reference path. MCTS is then employed to locally refine the path by exploring alternative route segments in the vicinity of obstacles. A modified Upper Confidence Bound (UCB) strategy is designed for path-level selection, enabling a balance between exploration of unexplored path segments and exploitation of promising partial paths with lower accumulated cost and collision risk. In addition, obstacle density information is incorporated to adaptively adjust the search behavior, improving robustness in complex environments.

**Experimental Results:** Experiments conducted in static grid-based environments with varying obstacle densities demonstrate that the proposed hybrid method achieves higher planning success rates (85–95%) than standalone A* or MCTS approaches. In complex multi-obstacle scenarios, the method produces paths that are approximately 15–25% shorter while maintaining improved planning efficiency compared with single-algorithm baselines.

**Limitations:** The approach is purely deliberative and does not incorporate learning mechanisms, which limits its ability to generalize across tasks or reuse experience from previous planning episodes. Moreover, the MCTS search tree must be reconstructed for each planning instance, resulting in rapidly increasing computational overhead in highly cluttered environments. The method is therefore less suitable for dynamic or large-scale scenarios where obstacle configurations change frequently.

**Difference from Proposed Algorithm:** In contrast to Zhang's MCTS–A* hybrid, which relies entirely on online search and handcrafted evaluation functions, the proposed method integrates A* guidance into deep reinforcement learning through reward shaping. This enables the agent to internalize heuristic information into a reusable policy rather than performing explicit tree search at every decision step. Furthermore, our framework naturally extends to multi-agent navigation under centralized training, whereas Zhang's method focuses exclusively on single-robot path planning.

### 2.7.3 A*-Guided Deep Reinforcement Learning

Lopez-Yepez et al. [4] proposed leveraging A* pathfinding to guide deep reinforcement learning in obstacle-dense environments. This work was presented at IEEE ICMLA 2024 and addresses the challenge of sparse reward problems in navigation tasks.

**Methodology:** The approach integrates A* pathfinding with deep reinforcement learning through an intrinsic reward mechanism. Specifically, A* computes optimal path distances from the agent's current position to the goal, and these distances are

used to shape the reward function. This provides dense feedback signals that guide the agent toward efficient navigation policies. The method was evaluated using three different DRL algorithms: Twin Delayed DDPG (TD3), Proximal Policy Optimization (PPO), and Rainbow DQN, allowing comparison across value-based and policy-gradient approaches.f

**Experimental Results:** Experiments conducted in obstacle-dense grid environments demonstrated that A*-guided reward shaping significantly accelerates policy learning convergence. All three tested algorithms showed improved training efficiency compared to sparse reward baselines. The agents achieved higher success rates and reduced collision frequencies, with Rainbow DQN showing particularly strong performance in complex obstacle configurations.

**Limitations:** The study is limited to single-agent scenarios with simplified 2D grid-based environment structures. The approach assumes static environments where A* can be computed offline, and does not address scenarios with dynamic obstacles. Additionally, the computational overhead of A* distance calculations during training is not extensively analyzed.

**Difference from Proposed Algorithm:** While Lopez-Yepez et al. demonstrate the effectiveness of A*-guided reward shaping for single-agent navigation, our proposed approach extends this concept in several important directions. First, the framework is expanded to multi-agent reinforcement learning scenarios, enabling the study of navigation tasks involving multiple agents within the same environment. Second, unlike Lopez-Yepez et al., who employ conventional MLP-based DRL architectures, this thesis also explores Transformer-based policy architecture to evaluate their suitability for learning navigation policies under A*-guided reward signals.

### 2.7.4 A*-Guided DRL for AGV Path Planning

Guo et al. [47] proposed an A*-guided deep reinforcement learning framework for efficient and adaptive automated guided vehicle (AGV) navigation in digital workshop environments.

**Methodology:** The framework integrates classical A* path planning with deep reinforcement learning through a reward shaping mechanism. Specifically, A* is used to generate reference paths and intermediate waypoints, from which distance-based intrinsic rewards are computed to guide the DRL agent toward efficient navigation behaviors. To handle dynamic environments, the method incorporates temporal information of moving obstacles, including other AGVs, workers, and mobile equipment, and predicts their future trajectories to enable proactive collision avoidance. The state representation combines lidar-based local occupancy maps, goal information, and predicted obstacle trajectories.

**Experimental Results:** Experiments conducted in simulated digital workshop environments demonstrate that the proposed A*-guided DRL approach reduces path length by approximately 25–30% compared to reactive DRL baselines without heuristic guidance. In addition, the method achieves around 40% faster training convergence and maintains high success rates exceeding 95% in scenarios involving up to 10 dynamic obstacles.

**Limitations:** The proposed framework focuses exclusively on single-AGV navigation and does not consider coordination or interaction among multiple agents. Although continuous A* re-planning enables adaptability in dynamic environments, it also introduces non-negligible computational overhead, which is not explicitly optimized. Furthermore, the potential challenges of integrating heuristic-based reward shaping into multi-agent reinforcement learning, such as non-stationarity and inter-agent reward interference, are not addressed.

**Difference from Proposed Algorithm:** While Guo et al. apply A*-guided reinforcement learning to single-agent industrial AGV navigation, the proposed method extends heuristic-guided learning to multi-agent reinforcement learning with explicit coordination mechanisms. Moreover, our evaluation emphasizes maze-like environments with complex and tightly constrained topologies, whereas Guo et al. focus on relatively open workshop layouts with sparse obstacles, allowing for a systematic analysis of A* guidance under structurally challenging navigation scenarios.

### 2.7.5 Multi-Agent DDPG with A* for Container Terminal Scheduling

Li et al. [48] proposed an integrated multi-agent deep deterministic policy gradient (MADDPG) framework combined with the A*algorithm to address multiple-equipment scheduling in automated container terminals.

**Methodology:** The study formulates a large-scale logistics optimization problem involving heterogeneous equipment, including quay cranes, yard cranes, and automated guided vehicles (AGVs). Each piece of equipment is modeled as an independent agent within a MADDPG framework to enable coordinated scheduling decisions. The A*algorithm is employed to generate collision-free routing plans for AGVs based on the terminal layout, ensuring path feasibility during task execution. The reward function jointly considers multiple operational objectives, such as task completion time, energy consumption, and equipment utilization. Centralized training with decentralized execution (CTDE) is adopted to mitigate partial observability and stabilize multi-agent learning.

**Experimental Results:** Simulation experiments based on real-world container terminal data demonstrate that the proposed A*-MADDPG approach reduces overall operational time by approximately 12–18% compared to conventional rule-based

scheduling strategies. In addition, energy consumption is reduced by around 15%, primarily due to improved AGV routing efficiency. The method exhibits stable learning performance in scenarios involving up to 20 simultaneously operating AGVs.

**Limitations:** The proposed framework is tailored to a domain-specific scheduling problem rather than general multi-agent navigation. The A*algorithm serves primarily as a path feasibility and routing module, without being integrated into the learning objective or reward structure of the agents. Furthermore, the method is evaluated exclusively using policy-gradient-based MARL and does not consider value-decomposition approaches such as QMIX, nor environments with complex maze-like spatial constraints.

**Difference from Proposed Algorithm:** While Li et al. integrate A* with MADDPG to support feasibility-aware scheduling in automated container terminals, the proposed method incorporates A*-derived distance information directly into the reward shaping mechanism to guide policy learning in navigation tasks. Moreover, our study systematically evaluates multiple MARL algorithms, including QMIX, MADDPG, and MAPPO, in maze-like environments with varying structural complexity, aiming to provide general insights into the compatibility between classical path planning heuristics and multi-agent reinforcement learning.

### 2.7.6 Hierarchical A*–DWA–DQN Path Planning

Zhang et al. [49] proposed a hierarchical path planning framework for mobile robots that integrates the A*algorithm, the Dynamic Window Approach (DWA), and Deep Q-Networks (DQN) to address navigation in complex environments.

**Methodology:** The framework adopts a three-layer architecture. At the global level, the A*algorithm generates a reference path composed of waypoints from the start to the goal. At the local level, DWA is responsible for real-time trajectory optimization and reactive obstacle avoidance. A DQN-based meta-controller operates at a higher decision level to adaptively switch between following the global A* path and activating DWA-based local avoidance based on the current environmental context. The DQN state representation includes local obstacle density, distance to the nearest A* waypoint, and robot velocity, while the action space consists of discrete mode-selection decisions, such as global path following, local reactive avoidance, or path re-planning.

**Experimental Results:** Experiments conducted in simulated complex environments, including narrow passages, U-shaped obstacles, and dynamic pedestrian scenarios, demonstrate that the integrated framework achieves a navigation success rate of approximately 92%, outperforming standalone A* (78%) and DWA-only (85%) baselines. Moreover, the learning-based mode-switching mechanism reduces unnecessary detours by around 20% compared to fixed rule-based switching strategies.

**Limitations:** The DQN component functions solely as a high-level mode selector and does not directly optimize navigation behavior or continuous control policies. As a result, the overall framework remains modular rather than end-to-end, limiting its ability to jointly optimize global planning and local control. In addition, the method is restricted to single-robot scenarios and relies on a discrete action space, without evaluation under continuous control settings or policy-gradient-based reinforcement learning methods.

**Difference from Proposed Algorithm:** While Zhang et al. employ DQN as a meta-controller to switch between independently designed planning and control modules, the proposed approach integrates A*-derived information directly into the learning objective through intrinsic reward shaping, enabling end-to-end policy learning for navigation. Furthermore, our framework extends heuristic-guided learning to multi-agent settings with continuous action spaces using MADDPG and systematically compares value-based (QMIX) and policy-gradient (MAPPO) multi-agent reinforcement learning algorithms.

### 2.7.7 Transformer-Based Heuristic Learning for Grid Pathfinding

Kirilenko et al. [21] introduced a hybrid framework called *TransPath*, a hybrid framework that combines neural prediction with classical search instead of replacing search entirely.

**Methodology:** The model predicts two map-conditioned guidance signals from occupancy grids: (i) a correction factor (CF) for heuristic calibration and (ii) a path probability map (PPM) for prioritizing promising states. A CNN–Transformer encoder-decoder produces these signals, which are then injected into weighted A* and Focal Search. This design preserves the reliability of symbolic planners while improving expansion order and search efficiency. Instead of replacing search, the model augments search guidance by improving node ordering and pruning ineffective expansions under non-uniform movement costs on 8-connected grids.

**Experimental Results:** On grid pathfinding benchmarks, TransPath substantially reduced node expansions (reported up to approximately 4x in difficult maps) while maintaining near-optimal path quality. It also showed a stronger efficiency-quality balance than several existing learning-based baselines.

**Limitations:** The work is restricted to single-agent static grids and supervised heuristic learning. It does not address MARL coordination, decentralized execution constraints, or dynamic obstacle adaptation.

**Relevance to My Thesis:** TransPath shows that integrating learned signals with classical search can significantly improve pathfinding efficiency while preserving the reliability of A*-based planners. However, the method is designed to enhance the

search process itself and remains limited to single-agent planning in static environments. This thesis extends the idea of leveraging A*-derived information beyond heuristic calibration by incorporating it into a reinforcement learning framework for navigation. Instead of guiding node expansion in a search algorithm, A*-generated information is used to shape intrinsic rewards during training under the centralized training with decentralized execution (CTDE) paradigm. This allows the learning agent to benefit from classical planning knowledge while developing policies that can operate in both single-agent and multi-agent navigation scenarios.

### 2.7.8 Search Dynamics Bootstrapping with Transformers

Lehnert et al. [22] introduced a Transformer-based framework called *Searchformer*, Searchformer learns to imitate A* search by representing search trajectories as token sequences. By modeling the reasoning process of classical search algorithms, the approach enables transformers to perform planning while progressively reducing the length of reasoning traces.

**Methodology:** Planning problems are represented as sequence data containing both task descriptions and A* execution traces. A Transformer is initially trained to imitate these search trajectories. The model is then iteratively fine-tuned on shorter valid traces generated by itself through a process called search dynamics bootstrapping. This training strategy progressively compresses reasoning trajectories while preserving solution quality in planning tasks such as maze navigation and Sokoban.

**Experimental Results:** The approach demonstrated strong generalization on unseen maze and Sokoban instances. Models trained with search-augmented traces achieved higher task-solving rates than solution-only sequence prediction, particularly under limited training data and increased task complexity. In addition, the search dynamics bootstrapping procedure often produced shorter reasoning traces than the original A* trajectories while preserving correct planning behavior.

**Limitations:** The approach is computationally intensive due to the generation and processing of long tokenized search traces. In addition, the method is primarily evaluated in synthetic single-agent planning environments such as maze navigation and Sokoban. It does not directly address continuous control settings, real-time multi-agent interaction, or decentralized decision-making commonly required in multi-agent reinforcement learning (MARL) systems.

**Relevance to My Thesis:** Searchformer proves that sequence modeling can represent not only the final solution path but also the latent reasoning process of A* search. However, its objective is to imitate and compress symbolic search traces within a token-based planning framework. In contrast, this thesis focuses on environment-interactive reinforcement learning for navigation. Rather than learning to reproduce

search trajectories, our approach incorporates A*-derived information as intrinsic reward signals to guide policy optimization during training. This shifts the role of A* from a sequence prediction target to a source of structured guidance within the learning process, enabling direct integration with MARL algorithms such as QMIX, MADDPG, and MAPPO under the CTDE paradigm.

### 2.7.9 Transformer-Driven Visual Intelligence for UAV Path Optimization

Jeevanandham et al. [23] proposed a Vision Transformer-based framework for aerial path optimization that models environments as grid or image representations, enabling route planning across heterogeneous terrains.

**Methodology:** The framework employs Vision Transformer architecture with patch embeddings and multi-head self-attention to capture both global and local spatial dependencies from grid or image-based environment representations. Path generation is formulated as a multi-objective optimization problem that balances travel distance, estimated energy consumption, and trajectory smoothness. The proposed method is evaluated against classical path planning algorithms, including A*, Dijkstra, and Greedy BFS, as well as CNN-based approaches.

**Experimental Results:** Experimental results in multi-terrain simulated environments—including urban, forest, desert, mountain, and coastal settings—show that the transformer-based approach produces shorter paths, lower energy consumption, and higher success rates than traditional baselines. The model also demonstrates improved robustness and decision efficiency in obstacle-dense and heterogeneous terrain scenarios.

**Limitations:** The evaluation is primarily conducted in single-UAV simulation environments and does not consider cooperative multi-UAV coordination or decentralized control constraints. In addition, the work does not integrate reinforcement learning or standardized multi-agent training frameworks such as CTDE-based MARL settings.

**Relevance to My Thesis:** This work highlights the effectiveness of attention-based architectures for modeling spatial structure in navigation tasks. However, its focus lies in improving a single planning model through transformer-based perception and multi-objective path optimization. In contrast, this thesis investigates how classical planning knowledge can be integrated into reinforcement learning algorithms. Rather than designing a new visual planner, our approach studies the interaction between A*-derived signals and different MARL algorithms by incorporating planner information as intrinsic rewards during policy optimization, enabling systematic comparison across algorithm families.

### 2.7.10 Summary of Recent Trends and Identified Gaps

Table 2.9 summarizes recent studies combining A* with learning-based planning methods from 2021 to 2025. The first six works mainly integrate A* with reinforcement learning or hybrid planning algorithms, where A* is used as a path reference, reward signal, feasibility check, or hierarchical planner. Most of these approaches focus on single-agent navigation, with limited exploration of multi-agent settings.

The last three works explore the use of transformer-based models for planning or heuristic learning. These studies demonstrate the potential of transformer architectures for predicting search dynamics or visual path planning, but they remain limited to single-agent tasks. Overall, existing works either focus on A*-guided learning or transformer-based planning, while the integration of transformer models with A*-guided multi-agent reinforcement learning remains relatively unexplored.

Table 2.9: Summary of Recent A*-Integrated Learning Approaches (2021–2025)

| Study | Year | Planning (Training) | Planning (Execution) | Learning | Multi-Agent | Environment | Key Contribution |
|---|---|---|---|---|---|---|---|
| Jin et al. | 2021 | A* | A* | PPO | No | Simple obstacles | Sequential A*-guided PPO refinement |
| Zhang H.Q. | 2024 | A* + MCTS | A* | – | No | Multi-obstacle | Hybrid planning optimization |
| Lopez-Yepez et al. | 2024 | A* distance | – | TD3/PPO/Rainbow | No | Dense obstacles | A*-based intrinsic reward shaping |
| Guo et al. | 2025 | A* | – | DRL | No | Digital workshop | Dynamic obstacle prediction |
| Li et al. | 2025 | A* | A* | MADDPG | Yes | Container terminal | Path feasibility checking |
| Zhang et al. | 2025 | A* + DWA | A* + DWA | DQN | No | Complex static | Hierarchical planner switching |
| Kirilenko et al. | 2022 | A* search traces | – | CNN + Transformer | No | Grid pathfinding | Learning heuristic proxies for search |
| Lehnert et al. | 2024 | A* traces | – | Transformer | No | Maze/Sokoban | Search dynamics imitation |
| Jeevanandham et al. | 2025 | Reference planner | – | Vision Transformer | No | Multi-terrain UAV | Transformer-based path optimization |

Table 2.10 further compares several representative A*-integrated learning approaches with the proposed method. Only closely related works that combine A* with reinforcement learning are included in this comparison.

Table 2.10: Comparison of Representative A*-Integrated Learning Methods and the Proposed Approach

| Aspect | Jin | Zhang H.Q. | Lopez-Yepez | Guo | Li | Zhang | Proposed |
|---|---|---|---|---|---|---|---|
| A* Integration Method | Path reference | Hybrid search | Intrinsic reward | Reward shaping | Feasibility check | Mode selection | Intrinsic reward |
| Agent Setting | Single-agent | Single-agent | Single-agent | Single-agent | Multi-agent | Single-agent | Single-agent / Multi-agent |
| Learning Algorithms | PPO | – | TD3 / PPO / Rainbow | DRL | MADDPG | DQN | QMIX / MADDPG / MAPPO |
| Environment Complexity | Low | Medium | Low–Medium | Medium | Domain-specific | Medium | Medium –High |
| Architecture Comparison | No | No | MLP only | No | No | No | MLP vs Transformer |
| Scalability Analysis | No | No | No | Limited | Limited | No | Yes |

**Key Observations from Recent Literature:**

1. **Limited Multi-Agent Investigation:** Among the reviewed works, only Li et al. [48] address multi-agent scenarios, and their focus is on domain-specific container terminal scheduling rather than general multi-agent navigation. The potential of A*-guided learning for multi-agent coordination in navigation tasks remains largely unexplored.

2. **Diverse A* Integration Strategies:** The reviewed works employ different strategies for integrating A* with learning: sequential path refinement (Jin et

al. [45]), intrinsic reward shaping (Lopez-Yepez et al. [4], Guo et al. [47]), hybrid search-based planning (Zhang H.Q. [46]), path feasibility checking (Li et al. [48]), and hierarchical mode selection (Zhang et al. [49]). However, systematic comparison of these integration strategies is absent from the literature.

3. **Environment Complexity Limitations:** Most studies use relatively simple environments. Jin et al. use simple obstacle configurations, Lopez-Yepez et al. focus on dense but structurally simple grids, while Guo et al. target open warehouse layouts. Complex topologies such as maze-like structures, narrow corridors, bottlenecks, and multi-room layouts have not been systematically studied with A*-guided learning approaches.

4. **Algorithm Coverage Gap:** No existing work provides comprehensive evaluation across multiple MARL paradigms. Lopez-Yepez et al. compare TD3, PPO, and Rainbow DQN in single-agent settings, but value decomposition methods (VDN, QMIX) and recent multi-agent policy gradient methods (MAPPO) have not been evaluated with A*-guided learning.

5. **Architecture Exploration:** Existing A*-integrated reinforcement learning studies primarily rely on MLP-based policy networks. Recent transformer-based planning approaches focus on heuristic learning or visual navigation but remain limited to single-agent settings. Consequently, the role of transformer architectures in A*-guided multi-agent reinforcement learning has not been systematically investigated.

6. **Static vs. Dynamic Environment Trade-offs:** While Guo et al. [47] address dynamic obstacles through prediction, and Zhang H.Q. [46] focuses on static multi-obstacle environments, the computational trade-offs of A* re-planning in dynamic multi-agent scenarios have not been thoroughly analyzed.

These observations motivate the present thesis, which aims to address these gaps through a systematic investigation of A*-guided reinforcement learning in multi-agent navigation scenarios. Specifically, this work evaluates multiple MARL algorithms ( QMIX, MADDPG, MAPPO) with A*-based intrinsic rewards in complex environments, while also comparing MLP and Transformer-based policy architectures for improved multi-agent coordination.

## 2.8 Research Gaps

Based on the comprehensive review of recent literature presented above, several important research gaps have been identified in the integration of classical path planning

algorithms with deep reinforcement learning (DRL). While works such as Jin et al. [45], Lopez-Yepez et al. [4], Guo et al. [47], Li et al. [48], Zhang H.Q. [46], and Zhang et al. [49] have made valuable contributions to A*-integrated learning approaches, recent studies such as Kirilenko et al. [21], Lehnert et al. [22], and Jeevanandham et al. [23] have explored transformer-based architectures for heuristic learning and visual path planning. However, significant limitations remain. These limitations, synthesized from the reviewed individual studies, form the basis of the research gaps addressed in this thesis.

### 2.8.1 Limitation to Single-Agent Scenarios

Among the nine reviewed works, the majority focus exclusively on single-agent navigation scenarios. Only one study considers a multi-agent setting, but it is restricted to a domain-specific industrial application and does not address general multi-agent navigation problems. Moreover, none of the existing studies systematically investigate multi-agent coordination, conflict resolution, or learning stability under the centralized training with decentralized execution (CTDE) paradigm. As a result, it remains unclear whether A*-guided learning can effectively improve learning efficiency and cooperation in general multi-agent reinforcement learning (MARL) navigation tasks.

### 2.8.2 Simplified Environment Structure

The environments used in the reviewed works exhibit limited topological complexity. Jin et al. [45] use simple obstacle configurations, Lopez-Yepez et al. [4] focus on dense but structurally simple grid environments, Guo et al. [47] target open warehouse layouts with sparse obstacles, Zhang H.Q. [46] addresses multi-obstacle but grid-based environments, Li et al. [48] operate in domain-specific container terminal layouts, and Zhang et al. [49] evaluate on medium-complexity static environments. None of these works include maze-like layouts, narrow corridors, bottlenecks, or enclosed regions, which are common in realistic navigation tasks and substantially increase difficulty. Thus, the effectiveness of A*-guided learning in structurally complex environments with challenging topologies remains unexplored.

### 2.8.3 Limited Exploration of Reward Shaping

The reviewed works employ diverse A* integration strategies: sequential path refinement [45], intrinsic reward shaping [4, 47], hybrid search-based planning [46], path feasibility checking [48], and hierarchical mode selection [49]. However, these approaches primarily rely on using A*-generated distances or waypoints as reward signals without systematic comparison of their effectiveness. None of the works examine potential issues

such as misleading guidance in structurally ambiguous regions, dynamic adjustment of guiding signals in multi-stage tasks, or planner-induced conflicts when multiple agents receive conflicting A* guidance. Therefore, the generality and robustness of different A* reward integration strategies remain insufficiently understood.

### 2.8.4 Lack of Cross-Algorithm Evaluation

Existing studies evaluate A*-integrated learning methods on a limited set of reinforcement learning algorithms. Most works focus on a single algorithm or a small group of algorithms within single-agent settings, and no study provides a comprehensive evaluation across multiple multi-agent reinforcement learning (MARL) paradigms. In particular, value decomposition methods such as QMIX and recent multi-agent policy gradient approaches such as MAPPO have not been systematically evaluated within A*-guided learning frameworks.

In addition, most A*-integrated reinforcement learning approaches employ conventional multilayer perceptron (MLP) policy networks. Although recent studies have explored transformer-based architectures for heuristic prediction or visual path planning, these approaches remain limited to single-agent planning tasks and are not integrated with A*-guided MARL learning. Consequently, it remains unclear whether A*-guided frameworks generalize effectively across different learning algorithms, policy architectures, and MARL paradigms.

### 2.8.5 Absence of Scalability and Computational Analysis

The computational cost of incorporating A* may become significant in high-dimensional or multi-agent environments. Among the reviewed works, Guo et al. [47] acknowledge the overhead of continuous A* re-planning but do not fully optimize it, Zhang H.Q. [46] notes increased computational cost in dense obstacle configurations, and Li et al. [48] provide limited scalability analysis for their multi-equipment scheduling scenario. None of the works systematically analyze the scalability of A*-integrated learning frameworks with respect to the number of agents, environment size, state dimensionality, or real-time constraints. This leaves a significant gap in understanding the practical feasibility of A*-guided approaches for large-scale multi-agent navigation.

### 2.8.6 Summary

Based on the analysis of recent A*-integrated learning studies, several important research gaps can be identified:

1. Limited investigation of A*-guided reinforcement learning in general multi-agent navigation scenarios; existing multi-agent studies are often domain-specific and

do not address general coordination and conflict resolution.

2. Evaluation in structurally complex environments remains scarce, particularly in maze-like layouts with narrow passages and constrained topologies.

3. Lack of comprehensive comparisons across representative multi-agent reinforcement learning algorithms, including value-decomposition methods (e.g., QMIX) and policy-gradient approaches (e.g., MADDPG, MAPPO).

4. Insufficient exploration of modern attention-based architectures, such as Transformers, for processing heuristic-guided information and supporting coordination in multi-agent settings.

5. Limited analysis of scalability and computational costs with increasing numbers of agents and environment complexity.

To address these gaps, the present thesis proposes a unified experimental framework that integrates A*-based intrinsic reward To address these gaps, this thesis proposes a unified experimental framework that integrates A*-based intrinsic reward shaping with multiple multi-agent reinforcement learning (MARL) algorithms. The framework systematically evaluates both MLP and Transformer-based policy architectures, and analyzes learning performance, scalability, and generalization in complex navigation environments. Although MARL algorithms are considered throughout, controlled single-agent experiments are first conducted to isolate the effects of planning guidance, followed by an extension to genuine multi-agent scenarios.

# Chapter 3

# Methodology

## 3.1 Overview

This chapter presents the methodology for integrating A* pathfinding with deep reinforcement learning using multi-agent algorithms. While MARL algorithms are employed throughout, the primary experimental focus is on controlled single-agent navigation to isolate the effects of planning guidance, with multi-agent scenarios explored as an extension. The research follows a five-stage progression:

1. **Baseline Reproduction and Validation:** Reproduce and validate the A*-DRL integration approach proposed by Lopez-Yepez et al. [4] by conducting experiments in single-agent settings within the multi-agent frameworks VMAS and BenchMARL.

2. **Cross-Algorithm Evaluation of A*-Guided Learning:** Evaluate the A*-DRL hybrid approach under representative multi-agent reinforcement learning algorithms (MAPPO, QMIX, and MADDPG) in single-agent environments, where these algorithms reduce to their single-agent formulations.

3. **Design and Evaluation of the Task-Adaptive Waypoint-Aware Transformer (TA-WAT) Configuration:** Design and evaluate a Transformer-based policy configuration that incorporates A*-derived waypoint observations while retaining the same core architecture and training settings as the Standard Transformer baseline.

4. **Scaling A*-Guided Learning from Single-Agent to Dual-Agent Navigation:** Extend the A*-guided learning framework to cooperative dual-agent navigation and evaluate whether the benefits observed in single-agent settings are maintained under Dual-agent interactions.

5. **Extension of TA-WAT from Single-Agent to Dual-Agent Navigation:** Extend the Task-Adaptive Waypoint-Aware Transformer (TA-WAT) policy to cooperative dual-agent navigation and examine its observed behaviour under increased environmental and coordination complexity.

The system architecture integrating these components is illustrated in Figure 3.1.

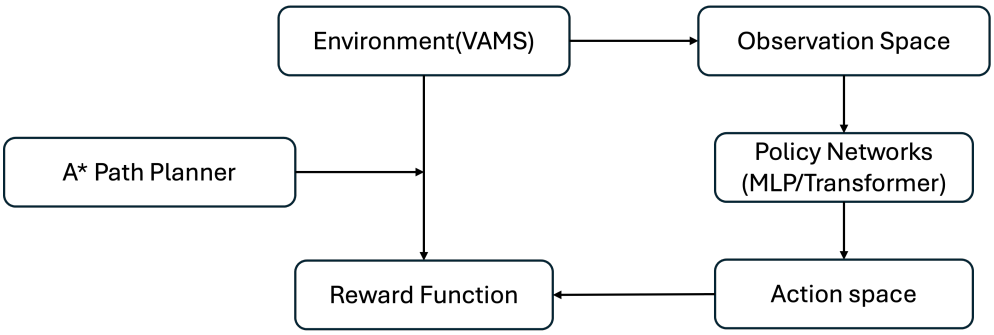



Figure 3.1: System architecture showing the integration of A* pathfinding with MARL training pipeline.

## 3.2 Simulation Platform

### 3.2.1 VMAS and BenchMARL

VMAS is a lightweight, highly configurable multi-agent simulation platform designed for fast, scalable experimentation. Its vectorized environment architecture enables a large number of parallel simulations, allowing efficient data collection and significantly reducing training time. Thanks to its modular structure and flexible scene configuration, VMAS supports a wide range of cooperative and competitive multi-agent tasks, making it well-suited for navigation, obstacle avoidance, and coordination studies. Figure 3.2 shows an example VMAS environment with four navigating agents.

BenchMARL serves as a unified training and evaluation framework built on top of VMAS, an environment for multi-agent scenarios. It provides standardized implementations of common algorithms for multi-agent reinforcement learning (MARL) and consistent interfaces for how agents receive information (observations) and make decisions (actions). The framework also offers a reproducible process for setting up experiments. By enforcing uniform training and evaluation procedures across algorithms, BenchMARL ensures fair comparisons, reduces the burden of coding new implementations, and improves reliability.

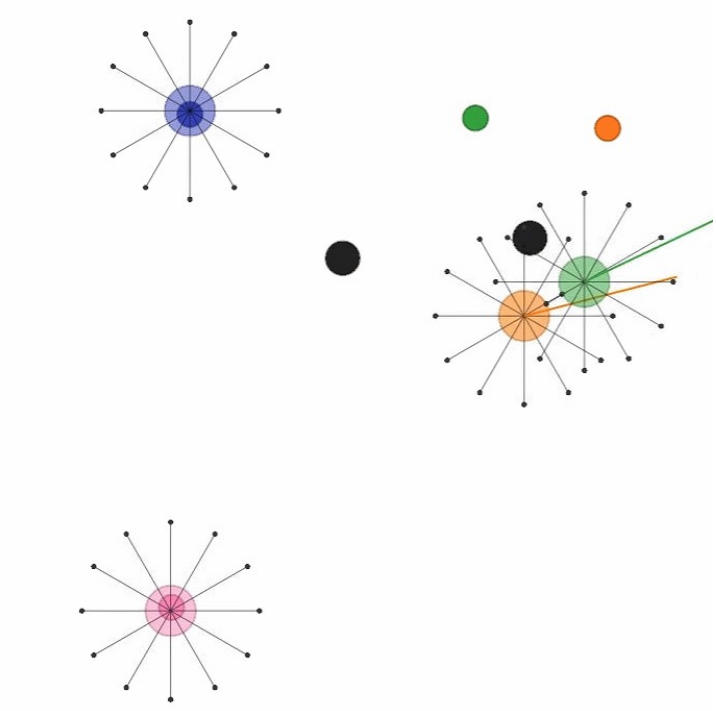


Figure 3.2: VMAS simulation environment with four navigating agents.

### 3.2.2 Environment Specification

The navigation environment is configured as a bounded 2D world with the specifications shown in Table 3.1.

Table 3.1: Environment and Agent Specifications

| Category | Parameter | Value | Description |
|---|---|---|---|
| World | Size | $10 \times 10$ units | Navigable area |
| | Grid Resolution | $50 \times 50$ cells | A* planning grid |
| | Max Episode Steps | 500 | Termination limit |
| Agent | Radius | 0.1–0.25 units | Collision detection |
| | Action Range | $[-2.0, 2.0]$ | Force per axis |
| | Agent Count | 1–10 | Configurable |
| LiDAR | Range | 1.0–3.0 units | Detection distance |
| | Rays | 16 | Angular resolution |
| | Field of View | 360° | Full surround |

Both continuous and discrete action spaces were considered in the environment design, with detailed information provided in Sections 3.4.3 and 3.4.4.

### 3.2.3 Map Difficulty Levels

This experiment uses a fully enclosed experimental environment. The maps are divided into three difficulty levels. Details can be seen in Table 3.2.

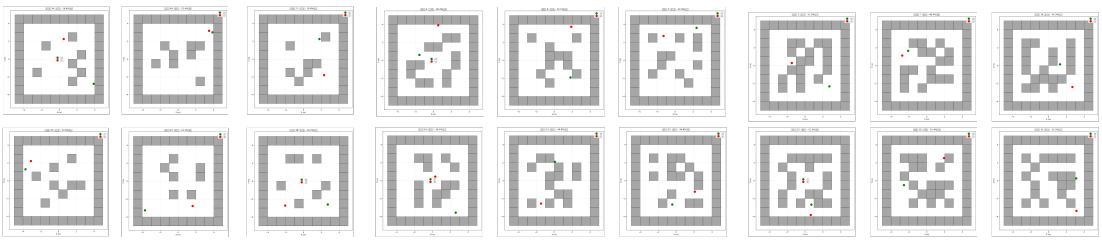
Table 3.2: Map Difficulty Characteristics

| Level | Obstacle Density | Navigation Challenge |
|---|---|---|
| Easy | Low (15–25%) | Simple paths |
| Medium | Moderate (30–40%) | Requires basic planning |
| Hard | High (45–55%) | Narrow passages |

The complete map archive contains 150 pre-generated maps, with 50 maps for each difficulty level (Easy, Medium, and Hard). However, all experiments reported in this thesis use only the 50 Easy maps: maps 1–30 are used for training, and maps 31–50 are reserved for testing. The Medium and Hard subsets are retained for future evaluation and are not used in the reported experiments. The map definitions, metadata, and experimental split are included in the accompanying project archive. Table 3.3 summarises the map allocation. Figure 3.3 presents representative maps; the Easy, Medium, and Hard examples are identified in Figures 3.3a, 3.3b, and 3.3c, respectively.

Table 3.3: Map collection and experimental split.

| Difficulty | Archive Total | Training Use | Testing Use |
|---|---|---|---|
| Easy | 50 | 1–30 | 31–50 |
| Medium | 50 | Not used | Not used |
| Hard | 50 | Not used | Not used |
| Reported experiments | 50 | 30 | 20 |



(a) Easy (b) Medium (c) Hard

Figure 3.3: Sample maps from each difficulty level (6 examples per level from 50 total).

All experiments presented below are conducted based on this experimental map configuration.

## 3.3 Policy Network Architectures

All architectures in this work follow the **actor-critic paradigm**, which serves as the foundational framework for both Fully Connected MLP Policy Baseline and Transformer-based policies. We first introduce the actor-critic framework and its mathematical formulation, then detail each specific architecture.

### 3.3.1 Actor-Critic Framework

The actor–critic framework is a fundamental architecture in reinforcement learning that combines two neural networks working in tandem. Its overall structure is illustrated in Figure 3.4.

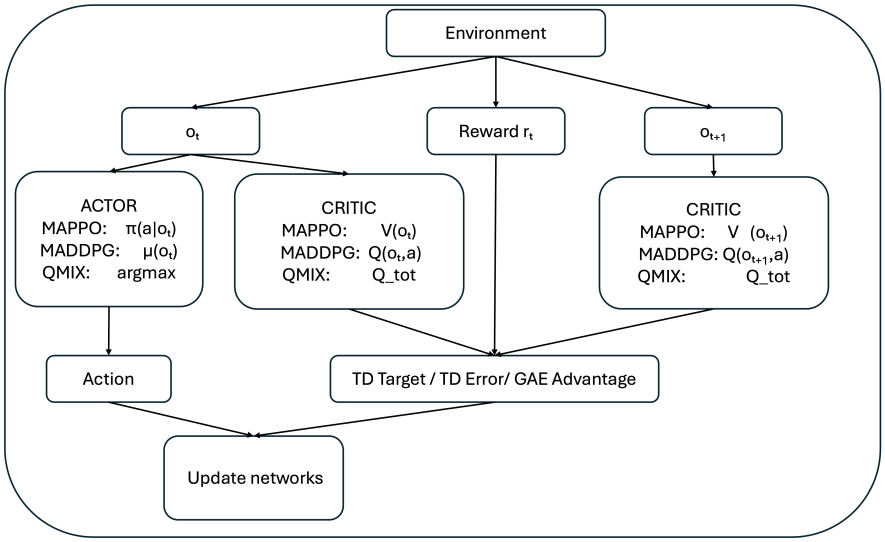



Figure 3.4: Actor-Critic Framework.

Algorithm 7 is an author-synthesized comparative summary based on the standard formulations of MAPPO, MADDPG, and QMIX [16, 15, 14]. It illustrates how each algorithm implements the interaction and update steps differently and does not represent a new learning algorithm proposed in this thesis.

- **Actor**: Maps observations to actions. In actor–critic methods (e.g., MAPPO and MADDPG), this corresponds to an explicit policy network. In value-based methods such as QMIX, actions are selected implicitly by maximizing value estimates, without a separate actor network.
- **Critic**: Estimates value functions to provide learning signals for policy optimisation. Its specific form depends on the underlying algorithm, such as state-value or action-value functions, and may be centralized or decentralized.

Table 3.4 summarizes the implementation differences across algorithms.

**Algorithm 7** Author-Synthesized Interaction and Update Summary for MAPPO, MADDPG, and QMIX [16, 15, 14]

**Require:** Actor network $\pi_\theta$ (or $\mu_\theta$), Critic network $V_\phi$ (or $Q_\phi$)
**Ensure:** Updated networks
Observe current state $o_t$
▷ Decision Step
Generate action using decision network:
    MAPPO: $a_t \sim \pi_\theta(a|o_t)$ ▷ Sample from Gaussian
    MADDPG: $a_t \leftarrow \mu_\theta(o_t)$ ▷ Deterministic output
    QMIX: $a_t \leftarrow \arg\max_a Q(o_t, a)$ ▷ Greedy selection
Execute $a_t$, receive reward $r_t$, next observation $o_{t+1}$
▷ Evaluation Step
Compute value estimates:
    MAPPO: $V(o_t)$, $V(o_{t+1})$
    MADDPG: $Q(o_t, a_t)$, $Q(o_{t+1}, \mu(o_{t+1}))$
    QMIX: $Q_{tot}(s_t, a_t)$, $\max Q_{tot}(s_{t+1})$
▷ Update Step
Compute TD target: $y_t \leftarrow r_t + \gamma \cdot \text{NextValue}$
Compute TD error: $\delta_t \leftarrow y_t - \text{CurrentValue}$
Update networks using $\delta_t$

Table 3.4: Network Components Across Algorithms

| **Component** | **MAPPO** | **MADDPG** | **QMIX** |
|---|---|---|---|
| Decision Type | Stochastic | Deterministic | Implicit |
| Action Generation | $a \sim \mathcal{N}(\mu, \sigma^2)$ | $a = \mu(o)$ | $a = \arg\max Q$ |
| Evaluation Type | State Value | Action Value | Factored Value |
| Value Output | $V(o)$ | $Q(o, a)$ | $Q_{tot}$ |

### 3.3.2 Fully Connected MLP Policy Baseline

The Fully Connected MLP Policy Baseline serves as the baseline architecture, implementing a standard actor-critic structure:

$$\begin{aligned}\text{Actor:}\quad & \text{obs} \to \text{FC}(256) \to \text{Tanh} \to \text{FC}(256) \to \text{Tanh} \to \text{FC}(128) \to \text{action} \\ \text{Critic:}\quad & \text{obs} \to \text{FC}(256) \to \text{Tanh} \to \text{FC}(256) \to \text{Tanh} \to \text{FC}(128) \to \text{value}\end{aligned}$$

- **Architecture**: Three hidden layers with dimensions [256, 256, 128].
- **Activation**: Tanh activation for bounded outputs.
- **Parameter sharing**: Separate networks for actor and critic.
- **Input**: Flattened observation vector (42 dimensions for standard observation), composed of agent state (4D), goal-relative position (2D), obstacle-relative positions for 10 obstacles (20D), and 16 LiDAR measurements, as detailed in Table 3.6.

### 3.3.3 Standard Transformer Policy Baseline

The Standard Transformer Policy Baseline uses self-attention to model relationships across agents at the current timestep. For each agent, the complete observation vector—including its state, goal information, other-agent features, obstacle features, LiDAR measurements, global context, and the shared waypoint field—is projected into one fixed-size embedding. Self-attention then operates across the agent dimension; goal, obstacle, LiDAR, and waypoint features are contained within each agent embedding rather than represented as separate tokens. As shown in Figure 3.5, the resulting agent embeddings pass through the Transformer encoder before the policy produces actions.

Key design choices include:

- **Input projection**: Each agent's complete observation vector is projected into a fixed-size embedding of dimension $d_{model}$.
- **Spatial encoding**: The agent's normalized two-dimensional position is mapped through sinusoidal functions with learned frequency bands and added to its projected observation embedding.
- **Agent-slot encoding**: A sinusoidal encoding of the agent's input-slot index is also added. This distinguishes agent slots and represents input order, not temporal order.

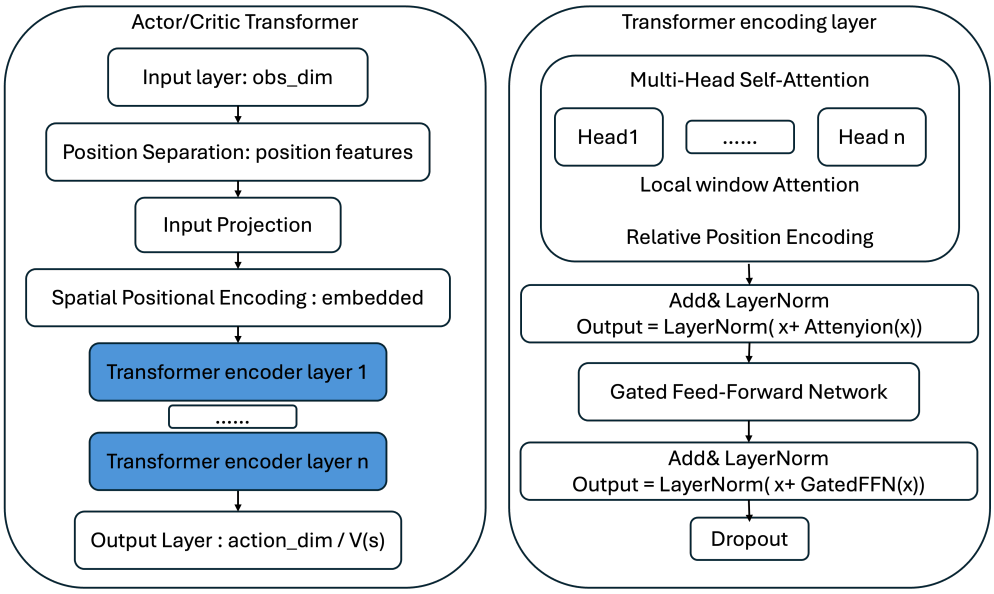



Figure 3.5: Transformer policy network architecture with per-agent observation projection, positional encoding, and self-attention across agents.

- **Relative positional bias**: Pairwise agent distance and direction are encoded and converted into a head-specific bias that is added to the attention logits. This mechanism is used in both Transformer variants.
- **Self-attention**: Multi-head self-attention captures relationships among agents while using the environmental information contained in each agent's observation vector.
- **Parallel processing**: Enables efficient batch computation on GPU.

Positional information is needed because content-based self-attention does not by itself specify where an agent is located or how two agents are spatially arranged. In navigation, two agents can have similar observation content while requiring different actions because their absolute locations, separation distances, or relative directions differ. The implementation therefore combines content and spatial information. For agent $i$, the input to the first Transformer layer is

$$z_i = W_{in} o_i + E_{spatial}(p_i) + E_{slot}(i), \tag{3.1}$$

where $o_i$ is the complete observation vector, $p_i = (x_i, y_i)$ is the two-dimensional agent position, $W_{in}$ is the learned input projection, $E_{spatial}$ is the spatial position encoding, and $E_{slot}$ is the sinusoidal agent-slot encoding. The slot index identifies the agent's position in the input tensor and must not be interpreted as a timestep.

The relative-position mechanism then computes, for every pair of agents $(i, j)$,

$$\Delta p_{ij} = p_j - p_i, \qquad d_{ij} = \|\Delta p_{ij}\|_2, \qquad u_{ij} = \frac{\Delta p_{ij}}{d_{ij} + \epsilon}. \tag{3.2}$$

The distance $d_{ij}$ and direction $u_{ij}$ are mapped to a learned relative representation and projected to one bias value $b_{ij}^{(h)}$ for each attention head $h$. The attention computation is therefore

$$\text{Attention}^{(h)}(Q, K, V) = \text{softmax}\left(\frac{Q^{(h)}K^{(h)^T}}{\sqrt{d_k}} + B_{rel}^{(h)}\right)V^{(h)}, \tag{3.3}$$

where $Q$, $K$, and $V$ are query, key, and value matrices derived from the agent embeddings, $d_k$ is the key dimension used for scaling, and $B_{rel}^{(h)} = [b_{ij}^{(h)}]$ is the learned pairwise spatial bias for head $h$ [1]. Consequently, the attention weights depend on both observation content and inter-agent geometry.

The relative positional encoding represents relationships between agent positions. It is not an A*-waypoint-specific embedding and does not encode temporal history. In a single-agent environment, the only pairwise relation is the agent with itself, for which $\Delta p_{ii} = 0$; the inter-agent relative-position mechanism therefore degenerates and provides no non-trivial pairwise geometry. The same spatial, slot, and relative positional mechanisms are used by both the Standard Transformer and TA-WAT. They were not independently removed through component-level ablation, so the reported differences between the two variants cannot be attributed to positional encoding.

The Fully Connected MLP Policy Baseline serves as a lightweight and widely adopted baseline. It processes observations through stacked fully connected layers, providing efficient function approximation while assuming a fixed and implicit structure in the input. This design is effective when the observation space is relatively compact and agent interactions are limited.

In contrast, the Transformer-based policy is designed to capture relationships among agents represented at the current timestep. Environmental entities such as goals and obstacles influence attention through the features contained in each agent's observation embedding; they are not separate attention tokens in the implemented model. This architecture increases representational flexibility at the cost of additional computational complexity. Previous observations, actions, rewards, and recurrent hidden states are not included. Therefore, self-attention models relationships across current agent embeddings rather than providing temporal memory across timesteps.

### 3.3.4 Task-Adaptive Waypoint-Aware Transformer Policy

The term Task-Adaptive Waypoint-Aware Transformer (TA-WAT) is introduced in this thesis to describe the evaluated waypoint-aware configuration; it is not the

name of an existing technique adopted from previous work. TA-WAT is based on the Standard Transformer policy used in this study and differs intentionally only in the content of the shared five-dimensional waypoint field, which contains A*-derived values instead of zeros. The contribution therefore lies in the design and evaluation of this waypoint-input configuration rather than in proposing a fundamentally new Transformer architecture.

To investigate different levels of A* integration, two variants of the Transformer-based policy networks are defined:

- **Transformer (Baseline):**
  - A standard Transformer encoder is used as the policy network.
  - The policy uses the same relative positional encoding, local attention, gated feed-forward network, layer configuration, attention dimensions, and output structure as the waypoint-aware variant.
  - The shared five-slot waypoint field is zero-filled, so no A*-derived waypoint observation is available to the policy.
  - During evaluation, both A*-based reward shaping and A*-derived waypoint input are absent.
- **Waypoint-Aware Transformer:**
  - The core Transformer architecture and training parameters are identical to those of the Standard Transformer Policy Baseline.
  - The only intentional model-input difference is that the shared five-slot waypoint field contains the A*-derived values $(\Delta x, \Delta y, d, d_x, d_y)$ instead of zeros.
  - During training, waypoint information is available through both reward shaping and the observation vector.
  - During evaluation, A*-based reward shaping is disabled, but the five waypoint observation features remain available.

.

Table 3.5 summarizes the shared configuration and the waypoint-observation difference between the two Transformer variants.

## 3.4 Observation and Action Spaces

In reinforcement learning, an observation is the information available to an agent at a given time step, which may provide a complete or partial representation of the

Table 3.5: Comparison of Transformer Variants

| **Feature** | **Standard Transformer** | **TA-WAT** |
|---|---|---|
| A* Reward Shaping | Yes | Yes |
| Waypoint Field Values | Zero-filled | A*-derived |
| Gated Feed-Forward | Yes | Yes |
| Input Dimension | Identical | Identical |
| Output | Action distribution | Action distribution |

underlying environment state. Just as humans understand their surroundings through sight and hearing, our navigation agent needs to "see" nearby obstacles, goal positions, and other agents.

This research designed two observation spaces: one is standard observation, primarily used for Fully Connected MLP Policy Baseline, and the other is Transformer-enhanced observation, which supports attention mechanisms.

In reinforcement learning, an action is the choice the agent makes to affect its environment. For navigation tasks, actions determine how the agent moves within the environment. This framework supports two types of action spaces: continuous action spaces, where actions can take any value within a range (e.g., steering angle), and discrete action spaces, where a limited set of distinct actions is available (e.g., move forward, turn left, or turn right). These different spaces accommodate various reinforcement learning algorithms.

### 3.4.1 Standard Observation (MLP Policy Baseline)

In the Fully Connected MLP Policy Baseline, each agent receives a compact observation vector composed of essential navigation features:

1. Agent State (4D): Position $(x, y)$ and velocity $(v_x, v_y)$
2. Goal Information (2D): Relative position $(\Delta x_g, \Delta y_g)$
3. Other Agents (4D $\times n$): Relative position and velocity
4. Nearest Obstacles (2D $\times 10$): Relative positions
5. LiDAR (16D): Distance measurements from a 360° scan

This observation focuses on raw spatial coordinates without pre-computed distance or directional features. It is used by the Fully Connected MLP Policy Baseline and is distinct from the shared Transformer observation described below.

### 3.4.2 Shared Observation for Standard Transformer and TA-WAT

Both the Standard Transformer and TA-WAT receive the same 99-dimensional structured observation vector. Compared with the Fully Connected MLP Policy Baseline observation, this representation includes additional derived spatial features. The two Transformer variants differ only in the values of the shared five-slot waypoint field: it is zero-filled for the Standard Transformer and populated with A*-derived values for TA-WAT.

The Transformer-enhanced observation comprises the following components:

1. Agent Self-State (6D): Raw position $(x, y)$, normalized position $(x/W, y/W)$ with respect to world half-size $W$, and velocity $(v_x, v_y)$.

2. Goal Information (7D): Normalized goal position $(g_x/W, g_y/W)$, relative displacement $(\Delta x_g, \Delta y_g)$, Euclidean distance to goal $d_g$, and unit direction vector $(\Delta x_g/d_g, \Delta y_g/d_g)$.

3. Other Agents (5D $\times n$): For each of the $n$ nearest agents, relative position $(\Delta x_j, \Delta y_j)$, relative velocity $(\Delta v_{x,j}, \Delta v_{y,j})$, and inter-agent distance $d_j$. Missing slots are zero-padded.

4. Nearest Obstacles (6D $\times 10$): For the 10 closest obstacles, relative position $(\Delta x_o, \Delta y_o)$, distance $d_o$, obstacle size $s_o$, and unit direction vector $(\Delta x_o/d_o, \Delta y_o/d_o)$.

5. LiDAR (16D): Distance-based proximity measurements from 16 uniformly spaced rays covering 360°.

6. Global Map Context (5D): Normalized distances to the four environment boundaries (left, right, bottom, top) and local agent density within sensing range.

7. A* Waypoint Field (5D): Relative position to the next A*-planned waypoint $(\Delta x_w, \Delta y_w)$, waypoint distance $d_w$, and unit direction vector $(\Delta x_w/d_w, \Delta y_w/d_w)$. The field is zero-filled for the Standard Transformer and contains these A*-derived values for TA-WAT.

The shared Transformer observation has a larger input dimension than the MLP observation because it includes pre-computed normalized values, distances, and direction vectors. This larger representation is common to both Transformer variants and is not the variable isolated by the Standard Transformer–TA-WAT comparison. In reinforcement learning, an action is the choice the agent makes to affect its environment. For navigation tasks, actions determine how the agent moves within the environment. This framework supports two types of action spaces: continuous action spaces, where actions can take any value within a range (such as steering angle), and discrete action

Table 3.6: Comparison Between the MLP and Shared Transformer Observations

| Component | Standard (MLP) | Transformer | Key Enhancement |
|---|---|---|---|
| Agent State | 4D | 6D | Added normalized coordinates |
| Goal Information | 2D | 7D | Added distance and direction |
| Other Agents (0 agent) | 0 | 0 | Added inter-agent distance |
| Obstacles (10) | 20D | 60D | Added size and direction features |
| LiDAR | 16D | 16D | Unchanged |
| Global Context | – | 5D | Boundary proximity + density |
| A* Waypoint | – | 5D | Zero-filled / A*-derived |
| **Total Dimension** | **42D** | **99D** | +57D |

spaces, where a limited set of distinct actions is available (such as move forward, turn left, or turn right). These different spaces accommodate various reinforcement learning algorithms.

### 3.4.3 Continuous Action Space

MAPPO and MADDPG use a continuous action space. The agent outputs a 2D force vector applied to itself. It follows Newton's laws of motion: actions (forces) affect velocity and position through acceleration.

### 3.4.4 Discrete Action Space

The discrete action space is used in the QMIX algorithm, where the agent selects one action from a predefined finite set. In this study, the agent operates within a discrete action space comprising 9 distinct actions.

The two sets of action spaces are compared as shown in Table 3.7

Table 3.7: Comparison of Continuous and Discrete Action Spaces

| Property | Continuous | Discrete |
|---|---|---|
| Dimension | 2 | 1 |
| Value Range | $[-2.0, 2.0]$ per axis | 9 predefined directions |
| Precision | Infinite (any direction) | Limited (9 directions) |
| Algorithms | MAPPO, MADDPG | QMIX |
| Exploration | Gaussian noise | $\epsilon$-greedy |
| Trajectory | Smooth curves | Step-like patterns |

## 3.5 A* Pathfinding Integration

### 3.5.1 Overview

This section describes the integration of the A* pathfinding algorithm into the multi-agent reinforcement learning framework. It computes optimal paths on a discretized grid map. The algorithm provides waypoint-based navigation guidance to support policy learning. A* is used primarily as an auxiliary training mechanism for waypoint-based reward shaping. For the MLP baseline and Standard Transformer Policy Baseline, A*-derived guidance is removed during evaluation, and the learned policy acts using only the standard observation vector. However, for the Task-Adaptive Waypoint-Aware Transformer Policy, A*-based reward shaping is disabled during evaluation, but A*-derived waypoint observation features may remain in the observation vector. Therefore, only the non-waypoint-aware configurations can be described as fully mapless during evaluation. The integration of A* pathfinding addresses a primary challenge in obstacle-dense environments: the sparse reward problem, where agents receive feedback only upon reaching the goal. The integration architecture is shown in Figure 3.6.

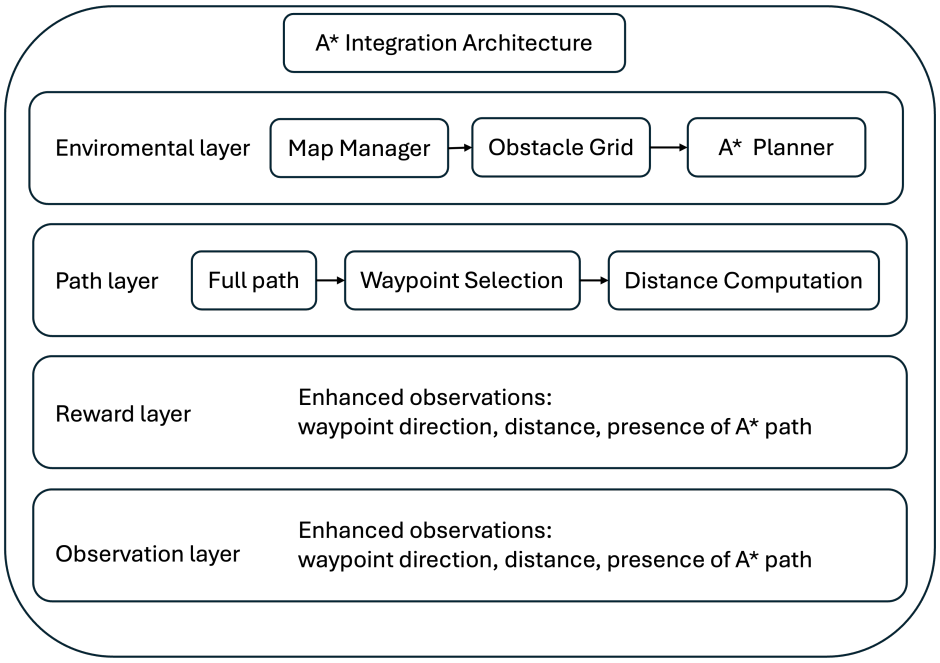



Figure 3.6: A* integration: in baseline configurations, waypoints guide training through reward shaping and are not used during evaluation; in Waypoint-Aware Transformer configurations, reward shaping is disabled during evaluation, but waypoint observation features may remain available to the policy.

### 3.5.2 Waypoint Selection Mechanism

A practical challenge in using A* guidance is selecting an appropriate waypoint from the computed path. During preliminary development, nearest-node selection was observed to produce frequent changes in the guidance target in some trajectories, particularly when path nodes were densely distributed or located near sharp turns. This was a qualitative development observation rather than the result of a controlled comparison.

Based on this development observation, a **look-ahead waypoint selection** mechanism was implemented. The mechanism first identifies the path node closest to the agent and then scans the subsequent nodes to select the first node whose Euclidean distance from the agent is at least $d_{\min}$. If no subsequent node satisfies this condition, the final path node is selected.

The threshold $d_{\min}$ defines the minimum distance between the agent and the selected look-ahead waypoint. A fixed value of $d_{\min} = 0.5$ environment units was used in all reported experiments. This value was selected during preliminary development to avoid selecting waypoints immediately adjacent to the agent while retaining relatively local path guidance. It was not determined through a systematic parameter search or a controlled ablation study and should therefore not be interpreted as an optimal value. This waypoint-selection threshold is separate from the task-completion condition, which is determined by whether the agent enters the goal region.

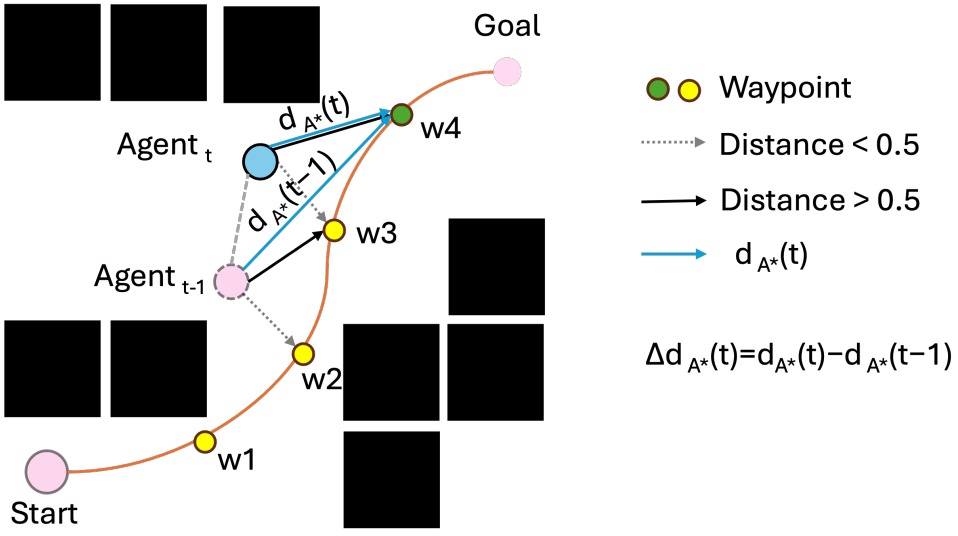



Figure 3.7: Illustration of the look-ahead waypoint selection rule. After identifying the nearest path node, the mechanism scans forward and selects the first subsequent node whose distance from the agent is at least $d_{\min}$. In the reported experiments, $d_{\min} = 0.5$ environment units.

Figure 3.7 illustrates the waypoint selection process. The agent identifies the closest point on the A* path and then scans forward to find the first subsequent node whose distance is at least $d_{\min}$. This rule was introduced to reduce frequent changes in the local guidance target. However, its effect on oscillation, trajectory smoothness, training efficiency, and final performance was not evaluated independently in a controlled ablation study.

### 3.5.3 Integration with Multi-Agent Framework

The A* pathfinding module is integrated into the VMAS + BenchMARL framework as follows:

1. Path Computation: At the beginning of each episode, A* paths are computed for all agents from their starting positions to their respective goals. Paths are recomputed when agents deviate significantly from the planned trajectory or when dynamic obstacles are detected.

2. Waypoint Update: At each timestep, the waypoint selection algorithm (Algorithm 8) is executed for each agent to determine the current guidance target.

**Algorithm 8** Look-Ahead Waypoint Selection Along A* Path

**Require:** Agent position $p$, A* path $P = [n_1, n_2, \ldots, n_k]$, look-ahead threshold $d_{\min}$
**Ensure:** Selected waypoint $w$

```
i* ← arg min_i ‖p − n_i‖                          ▷ Find closest path node
for j = i* + 1 to k do
    if ‖p − n_j‖ ≥ d_min then
        return n_j                                 ▷ First node beyond threshold
    end if
end for
return n_k                                         ▷ Default to goal if no node satisfies threshold
```

   The expected behaviour of the mechanism depends on the selected threshold. A smaller $d_{\min}$ generally permits the selection of waypoints closer to the agent and may result in more frequent changes in local guidance. A larger $d_{\min}$ generally selects more distant waypoints and may skip useful intermediate path information, particularly around turns and obstacles. These are mechanism-based expectations rather than experimentally established effects, because $d_{\min}$ was not varied in the reported experiments.

3. Reward Calculation: The A*-guided reward component is computed based on the agent's progress toward its current waypoint and added to the base reward signal.

4. Observation Augmentation: Optionally, the relative position to the current waypoint can be included in the agent's observation vector, providing explicit path information to the policy network.

Figure 3.8 shows an example of multi-agent navigation with A* guidance, where three agents navigate through a maze environment following their respective planned paths.

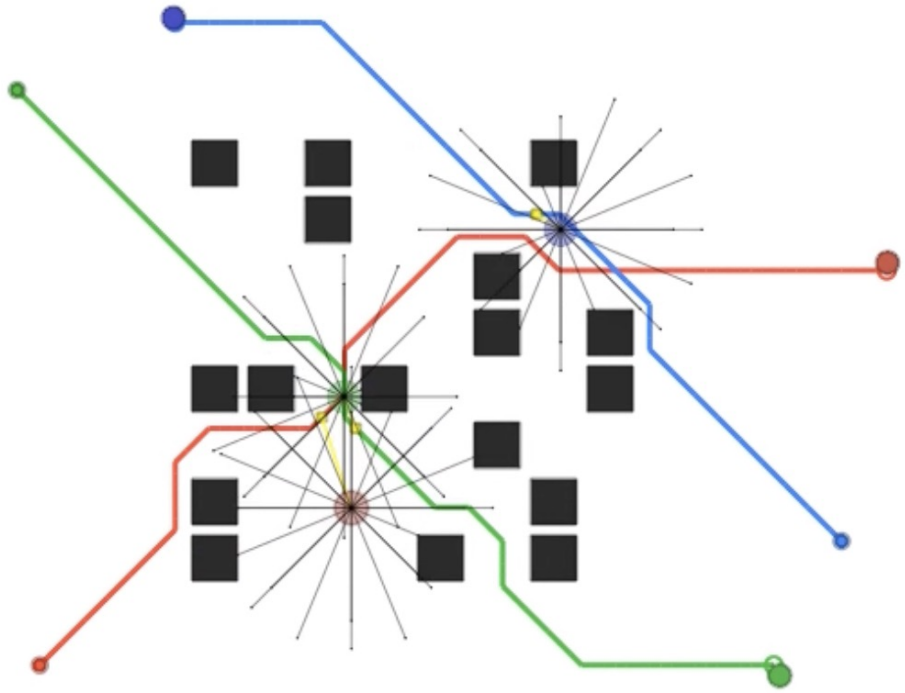
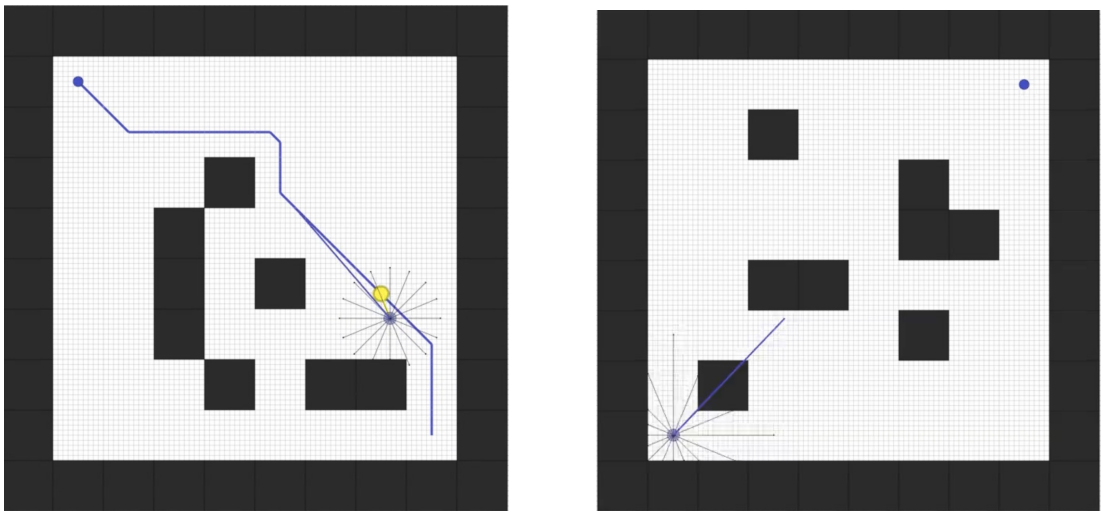


Figure 3.8: Example of multi-agent navigation with A* pathfinding guidance. Each agent follows its computed A* path toward its goal.

A runnable agent environment integrating A* planning is implemented. However, in practice, this experiment mainly used the method shown in Figure 3.9. This figure illustrates the training and evaluation environment.



(a) Training (b) Evaluation

Figure 3.9: Training and Evaluation in easy maps.

## 3.6 Reward Function

The reward function follows the design proposed in Lopez-Yepez et al. [4]. This work mainly evaluates two reward formulations: the Base Reward Function and the A*-Enhanced Reward Function. In addition, the Escape Reward Function was explored.

### 3.6.1 Base Reward Function

The base reward is defined as:

$$R(t) = -\lambda_g \, \Delta d_g(t) - C(t) + G(t),$$

where:

- $\Delta d_g(t)$ is the change in distance to the goal at timestep $t$,
- $C(t)$ is the collision penalty,
- $G(t)$ is a positive reward when the agent reaches a goal-related milestone.

### 3.6.2 A*-Enhanced Reward Function

To incorporate A* guidance, an additional shaping term is added based on waypoint progress in the A*-Enhanced Reward Function:

$$R(t) = -\lambda_g \, \Delta d_g(t) - C(t) + G(t) - \lambda_{A^*} \, \Delta d_{A^*}(t),$$

where:

- $\Delta d_{A^*}(t)$ is the change in distance to the current A* waypoint.
- $\lambda_{A^*}$ controls the strength of A*-based shaping.

## 3.7 Algorithm Configuration

Three MARL algorithms are evaluated: MAPPO (on-policy), QMIX (off-policy, discrete), and MADDPG (off-policy, continuous). Table 3.8 summarizes their key characteristics.

Table 3.8: Algorithm Comparison

| **Property** | **MAPPO** | **QMIX** | **MADDPG** |
|---|---|---|---|
| Policy Type | On-Policy | Off-Policy | Off-Policy |
| Action Space | Continuous | Discrete | Continuous |
| Value Decomposition | Shared Critic | Mixing Network | Centralized Critic |
| Exploration | Entropy Bonus | $\epsilon$-Greedy | Gaussian Noise |

Complete architectural equivalence is difficult to guarantee, and empirical instability is discussed in Chapter 4.

### 3.7.1 Hyperparameter Selection Strategy

Hyperparameters in this study were selected using a combination of automated baseline optimization and subsequent manual refinement. In the initial stage, Optuna was applied to MLP-based MAPPO models to identify stable baseline configurations under the specific training environment and reward structure used in this work. These Optuna-derived settings were fixed as reference parameters for subsequent experiments.

For longitudinal algorithm comparisons, each algorithm was evaluated using its own hyperparameter configuration, tuned under the same experimental protocol, following common practice in deep reinforcement learning. A shared hyperparameter configuration across different algorithms was not enforced, as this would unfairly disadvantage certain methods due to differences in training dynamics and parameter sensitivity.

When extending the policy architecture to Transformer-based models, automated hyperparameter optimization was no longer employed due to the higher computational cost and increased sensitivity of Transformer architectures. Instead, hyperparameters were manually adjusted based on the Optuna baseline and empirical observations of training stability, convergence behaviour, and performance consistency. For comparative evaluation, the Standard Transformer and TA-WAT were trained using the same network configuration, input dimension, and hyperparameter settings. The only intentional model-input difference was the content of the shared five-slot waypoint field: it was zero-filled for the Standard Transformer and contained A*-derived waypoint

values for TA-WAT. Therefore, observed differences are associated with the complete waypoint-input condition under the tested configuration. They should not be attributed separately to relative positional encoding, gated feed-forward networks, local attention, or other shared Transformer components.

### 3.7.2 Training Hyperparameters

The training configuration balances sample efficiency with computational resources. Key hyperparameters are shown in Table 3.9.

Table 3.9: Hyperparameters are based on Optuna-derived baseline configurations with limited empirical refinement.

| **Parameter** | **MLP** | **Transformer** |
|---|---|---|
| Batch Size (frames) | 60,000 | 60,000 |
| Parallel Environments | 120 | 120 |
| Minibatch Size | 4,096 | 4,096 |
| Minibatch Iterations | 30 | 15 |
| Learning Rate | $5 \times 10^{-4}$ | $1 \times 10^{-4}$ |
| Weight Decay | 0 | $1 \times 10^{-4}$ |
| Discount Factor ($\gamma$) | 0.99 | 0.99 |
| GAE $\lambda$ | 0.9 | 0.9 |
| Clip Epsilon | 0.2 | 0.15 |

Algorithm-specific settings that differ from this baseline are reported in the corresponding experimental chapters. The complete machine-readable configuration files are included in the accompanying project archive and document the optimizer settings, replay-buffer parameters, exploration schedules, update frequencies, target-network settings, and model parameters used for each reported configuration. Parameters not explicitly overridden use the defaults provided by BenchMARL 1.4.0.

During the exploratory phase of this study, several hyperparameters were adjusted to better understand the behavior of the learning algorithms under different settings. Based on these preliminary explorations, a stable baseline configuration was identified and adopted for all subsequent experiments. Unless otherwise specified, later experiments are conducted using this baseline, with only minor adjustments when required by changes in task complexity.

## 3.8 Performance Evaluation Metrics

The navigation environment generates instantaneous rewards, BenchMARL logs training and evaluation statistics, and project-specific scripts calculate the derived metrics reported in the experimental chapters. The following notation is used throughout

this section. Let $r_{e,t,a}$ denote the instantaneous reward received by agent $a$ at time step $t$ of episode $e$, where $t = 1, \ldots, T_e$, $T_e$ denotes the episode length, and $A$ denotes the number of agents. Let $R_k$ denote the mean episodic return logged at training iteration $k$, and let $E_j$ denote the mean evaluation return logged at evaluation point $j$.

### 3.8.1 Metrics Generated and Logged by BenchMARL

#### Instantaneous Environment Reward

The instantaneous reward is generated by the navigation environment rather than by BenchMARL itself. Under the basic reward configuration, the reward for agent $a$ at time step $t$ is

$$r_{t,a} = \lambda_d \left(d_{t-1,a} - d_{t,a}\right) + R_g \mathbb{I}\left(d_{t,a} < \rho_{g,a}\right) + P_c N_{t,a}^{\text{collision}} + w_A \left(q_{t-1,a} - q_{t,a}\right), \quad (3.4)$$

where $d_{t,a}$ is the distance between agent $a$ and its assigned goal, $\lambda_d$ is the goal-distance reward coefficient, $R_g$ is the goal-region reward, $\rho_{g,a}$ is the radius of the assigned goal, $P_c$ is the per-contact collision penalty, $N_{t,a}^{\text{collision}}$ is the number of collision contacts detected for agent $a$, $q_{t,a}$ is the distance to the current A* waypoint, and $w_A$ is the A*-based reward-shaping weight. A reward component is set to zero when the corresponding mechanism is disabled.

The first and final terms reward progress toward the goal and A* waypoint, respectively, while the negative $P_c$ penalizes contacts.

#### Episodic Return

BenchMARL uses the TorchRL `RewardSum` transform to accumulate the instantaneous environment rewards over an episode. The undiscounted episodic return of agent $a$ in episode $e$ is

$$G_{e,a} = \sum_{t=1}^{T_e} r_{e,t,a}, \quad (3.5)$$

where $G_{e,a}$ is an undiscounted return in reward units; the discount factor $\gamma$ used for optimization is not applied to this logged quantity.

#### Training Episode Reward

At training iteration $k$, BenchMARL reports the mean episodic return associated with episodes completed in the corresponding collection batch. The value is averaged over both agents and completed episodes:

$$R_k = \frac{1}{|\mathcal{D}_k|} \sum_{e \in \mathcal{D}_k} \left( \frac{1}{A} \sum_{a=1}^{A} G_{e,a} \right), \tag{3.6}$$

where $\mathcal{D}_k$ is the set of episodes completed during training iteration $k$, and $|\mathcal{D}_k|$ is the number of completed episodes. In a shared-reward configuration, $G_{e,a}$ represents the shared return supplied to the agents; otherwise it represents the individual return of agent $a$.

The value $R_k$ is recorded in the BenchMARL CSV logs under

`collection_agents_reward_episode_reward_mean`.

If no episode terminates within a collection batch, the corresponding episode-reward observation may be missing or undefined. Subsequent post-processing calculations therefore use only valid, non-missing observations.

### Evaluation Reward

At evaluation point $j$, BenchMARL runs $N_{\text{eval}}$ evaluation episodes. For each episode, rewards are summed over time and averaged across agents. The resulting evaluation reward is

$$E_j = \frac{1}{N_{\text{eval}}} \sum_{e=1}^{N_{\text{eval}}} \left[ \frac{1}{A} \sum_{a=1}^{A} \sum_{t=1}^{T_e} r_{e,t,a}^{\text{eval}} \right], \tag{3.7}$$

where $r_{e,t,a}^{\text{eval}}$ is the instantaneous reward received during evaluation. The value of $N_{\text{eval}}$ is experiment-specific and is reported in the corresponding experimental chapter.

The value $E_j$ is recorded under

`eval_agents_reward_episode_reward_mean`.

Evaluation reward is expressed in reward units. It is neither a percentage nor a direct count or proportion of successful episodes.

### Logged Collision Penalty

The navigation environment records the instantaneous collision penalty under `collision_rew`. At collection iteration $k$, BenchMARL averages this quantity over the agent-transition samples contained in the collection batch:

$$c_k = \frac{1}{M_k} \sum_{m=1}^{M_k} c_{k,m}, \tag{3.8}$$

where $c_{k,m}$ is the collision-penalty value of sample $m$, and $M_k$ is the number of agent-transition samples represented in collection iteration $k$. The resulting value is recorded under

`collection_agents_info_collision_rew`.

This logged statistic combines agent–obstacle and agent–agent collision penalties and does not distinguish between the two collision types.

### 3.8.2 Metrics Computed by the Post-Processing Scripts

The metrics described below are not generated directly by BenchMARL. They are calculated by the project-specific analysis scripts from the statistics stored in the BenchMARL CSV logs.

#### Average Training Reward

Let $K$ denote the number of valid `collection_agents_reward_episode_reward_mean` observations. The average training reward reported in the experimental tables is

$$R_{\text{avg}} = \frac{1}{K} \sum_{k=1}^{K} R_k. \tag{3.9}$$

This is an equal-weight mean of valid iteration-level log entries, not a pooled mean over all individual episodes.

#### Final Reward

The final reward is calculated as the mean of the last $L_R$ valid training-reward observations:

$$R_{\text{final}} = \frac{1}{L_R} \sum_{k=K-L_R+1}^{K} R_k, \qquad L_R = \min(100, K). \tag{3.10}$$

The reported final reward is therefore a window average rather than the last recorded value.

#### Final-Window Evaluation Reward

Let $J$ denote the number of valid `eval_agents_reward_episode_reward_mean` observations. The evaluation statistic reported in the main experimental tables is

$$E_{\text{final}} = \frac{1}{L_E} \sum_{j=J-L_E+1}^{J} E_j, \qquad L_E = \min(10, J). \tag{3.11}$$

This metric is termed the *final-window evaluation reward*. It is expressed in reward units and must not be interpreted as a percentage or as a directly observed binary success rate.

**Legacy Reward-Threshold Success Proxy**

An evaluation point is classified as successful when its mean evaluation return is positive:

$$S_j = \begin{cases} 1, & E_j > 0, \\ 0, & E_j \leq 0. \end{cases} \tag{3.12}$$

For $N$ valid evaluation points, the reward-threshold success proxy is defined as

$$\text{SuccessProxy} = \frac{1}{N}\sum_{j=1}^{N} S_j = \frac{1}{N}\sum_{j=1}^{N} \mathbb{I}(E_j > 0)\,. \tag{3.13}$$

This metric is the proportion of recorded evaluation points whose mean evaluation return is positive. It is therefore a reward-threshold success proxy based on recorded evaluation rewards, not the true environment-level task-completion success rate defined by the original all-goals-achieved condition.

For the five-seed analysis in Chapter 7, let $J_s$ be the number of valid evaluation observations for seed $s$ and define

$$L_s^{\text{stat}} = \max\left(1, \lfloor 0.1 J_s \rfloor\right). \tag{3.14}$$

The seed-level evaluation statistic is

$$E_s^{\text{stat}} = \frac{1}{L_s^{\text{stat}}} \sum_{j=J_s-L_s^{\text{stat}}+1}^{J_s} E_{s,j}. \tag{3.15}$$

The main tables use the final $\min(10, J)$ observations, whereas the five-seed tests use the final 10% for each seed.

For paired TA-WAT–Standard Transformer comparisons, define

$$D_s = E_{s,\text{TA-WAT}}^{\text{stat}} - E_{s,\text{Standard}}^{\text{stat}}. \tag{3.16}$$

For $S$ matched seeds, the paired $t$ statistic and Cohen's $d_z$ are

$$t = \frac{\bar{D}}{s_D/\sqrt{S}}, \qquad d_z = \frac{\bar{D}}{s_D}, \tag{3.17}$$

where $\bar{D} = S^{-1}\sum_{s=1}^{S} D_s$ and $s_D$ is the sample standard deviation of $D_s$. The three

algorithm-level $p$-values are Holm–Bonferroni adjusted at $\alpha = 0.05$.

### Collision-Penalty Proxy

Let $K_c$ denote the number of valid logged collision-penalty observations. The collision-penalty proxy is calculated as

$$C_{\text{proxy}} = \frac{1}{K_c} \sum_{k=1}^{K_c} |c_k|. \tag{3.18}$$

Lower values indicate a smaller mean penalty magnitude. Because $c_k$ is already an iteration-level average and contacts may be recorded more than once, $C_{\text{proxy}}$ is not a collision count, frequency, or probability.

### Reward Volatility

For the convergence and stability analyses, the rolling-window length is

$$w = \min\left(100, \left\lfloor \frac{K}{10} \right\rfloor\right). \tag{3.19}$$

The rolling reward volatility at training iteration $k$ is calculated as the sample standard deviation of the reward observations in the corresponding window:

$$\sigma_{\text{rolling}}(k) = \sqrt{\frac{1}{w-1} \sum_{i=k-w+1}^{k} \left(R_i - \mu_{R,k}^{(w)}\right)^2}, \tag{3.20}$$

where

$$\mu_{R,k}^{(w)} = \frac{1}{w} \sum_{i=k-w+1}^{k} R_i. \tag{3.21}$$

Lower rolling standard deviation indicates reduced short-term fluctuation in the training-reward trajectory.

### Convergence Smoothness

The training-reward trajectory is first smoothed using a rolling mean:

$$\bar{R}_k = \frac{1}{w} \sum_{i=k-w+1}^{k} R_i. \tag{3.22}$$

The absolute change between two consecutive smoothed reward values is

$$D_k = \left|\bar{R}_k - \bar{R}_{k-1}\right|. \tag{3.23}$$

For the convergence-smoothness curves, these absolute changes are themselves averaged over a rolling window:

$$S_k = \frac{1}{w} \sum_{i=k-w+1}^{k} D_i. \tag{3.24}$$

Lower $S_k$ indicates a smoother observed reward trajectory; it does not establish policy optimality.

### Late-Stage Reward Standard Deviation

Where late-stage reward variability is reported, the analysis uses the sample standard deviation of approximately the final 10% of the valid training-reward observations. The number of observations included is

$$L_{10} = \lceil 0.1K \rceil . \tag{3.25}$$

The late-stage reward standard deviation is then

$$\sigma_{\text{late}} = \sqrt{\frac{1}{L_{10} - 1} \sum_{k=K-L_{10}+1}^{K} (R_k - \mu_{\text{late}})^2}, \tag{3.26}$$

where

$$\mu_{\text{late}} = \frac{1}{L_{10}} \sum_{k=K-L_{10}+1}^{K} R_k. \tag{3.27}$$

This produces one late-stage summary value rather than a rolling curve.

### Coefficient of Variation

The arithmetic mean and sample standard deviation of the complete valid training-reward sequence are

$$\mu_R = \frac{1}{K} \sum_{k=1}^{K} R_k, \tag{3.28}$$

and

$$\sigma_R = \sqrt{\frac{1}{K - 1} \sum_{k=1}^{K} (R_k - \mu_R)^2}. \tag{3.29}$$

The coefficient of variation is then

$$\mathrm{CV} = \frac{\sigma_R}{|\mu_R|}. \tag{3.30}$$

Here, $\mu_R$ and $\sigma_R$ are the mean and sample standard deviation of the $K$ valid values of $R_k$. Lower CV indicates lower relative variability, but CV is unstable when $|\mu_R|$ is close to zero.

### Convergence Iteration

The convergence analysis first applies a 50-point rolling mean to the valid training-reward sequence:

$$\widetilde{R}_k^{(50)} = \frac{1}{\min(50, k)} \sum_{i=\max(1,k-49)}^{k} R_i. \tag{3.31}$$

The convergence iteration used by the project-specific analysis script is defined as the first training iteration at which this smoothed reward reaches 90% of the final reward:

$$k^* = \min \left\{ k : \widetilde{R}_k^{(50)} \geq 0.9 R_{\text{final}} \right\}. \tag{3.32}$$

This is a heuristic indicator, not proof of policy convergence, and is interpreted cautiously when $R_{\text{final}} \leq 0$.

The quantity $k^*$ is a training-iteration index rather than an environment-frame count. If each training iteration collects $B$ environment frames, the corresponding approximate frame count is

$$F^* = Bk^*. \tag{3.33}$$

### Training Time

Training time is obtained preferentially from the final applicable value of the cumulative BenchMARL timer:

$$T_{\text{train}} = \texttt{timers\_total\_time}\left(k_{\text{final}}\right). \tag{3.34}$$

This quantity represents the cumulative training-loop time recorded by BenchMARL for the included training iterations. If the BenchMARL timer is unavailable, the post-processing script estimates elapsed wall-clock time from the earliest and latest applicable result-file timestamps. Values obtained using this fallback procedure are approximate wall-clock estimates and are identified as such where relevant.

### 3.8.3 Metrics Not Directly Available from the Logs

#### Binary Success Rate

A binary episode-level success indicator can be defined as

$$S_e = \mathbb{I}\left(\bigcap_{a=1}^{A} \{d_{e,T_e,a} < \rho_{g,a}\}\right), \tag{3.35}$$

where $d_{e,T_e,a}$ is the final distance between agent $a$ and its assigned goal. Under this definition, a multi-agent episode is counted as successful only when all agents satisfy their respective goal-region conditions.

The corresponding binary success rate would be

$$\mathrm{SR} = \frac{1}{N_{\mathrm{eval}}} \sum_{e=1}^{N_{\mathrm{eval}}} S_e \times 100\%. \tag{3.36}$$

The environment implementation uses this all-agents goal-region condition as its task-completion termination condition. Nevertheless, the episode-level value of $S_e$ was not retained in the summary logs used for the reported experiments. Those logs also do not provide the episode-level termination information needed to distinguish successful goal completion from time-limit truncation. A direct binary success rate therefore cannot be reconstructed reliably and is not reported numerically. In particular, evaluation reward and the reward-threshold success proxy in Equation 3.13 must not be substituted for this measure or interpreted as the percentage of successful episodes.

#### Direct Collision Rate

A direct collision rate would require distinct collision-event counts normalized by episodes or environment steps. These counts were not logged; therefore, the collision-penalty proxy in Equation 3.18 is the only reported collision measure and must not be interpreted as a rate.

### 3.8.4 Relative Changes and Legacy Figure Labels

For a method value $E_m$ and baseline value $E_b$, the relative evaluation-reward change is

$$\Delta E_{\mathrm{rel}} = \frac{E_m - E_b}{|E_b|} \times 100\%. \tag{3.37}$$

This percentage describes a relative reward difference, not a binary success rate. Some embedded figures retain the legacy labels *Success Rate* and *Collision Rate* from

the original plotting scripts. These curves represent $E_j$ and $C_{\text{proxy}}$, respectively; the surrounding captions use the corrected terminology.

### 3.8.5 Metric Summary

Table 3.10 distinguishes the statistics logged directly by BenchMARL from those subsequently calculated by the project-specific post-processing scripts.

Table 3.10: Sources and Definitions of the Reported Performance Metrics

| Metric | Source Layer | Data Source | Desired | Unit or Interpretation |
|---|---|---|---|---|
| Training Episode Reward | BenchMARL | `collection_agents_reward_episode_reward_mean` | Higher | Reward units |
| Evaluation Reward | BenchMARL | `eval_agents_reward_episode_reward_mean` | Higher | Reward units |
| Logged Collision Penalty | BenchMARL | `collection_agents_info_collision_rew` | Closer to zero | Mean logged penalty per collection statistic |
| Average Training Reward | Post-processing | Mean of all valid $R_k$ | Higher | Reward units |
| Final Reward | Post-processing | Last $\min(100, K)$ values of $R_k$ | Higher | Reward units |
| Final-Window Evaluation Reward | Post-processing | Last $\min(10, J)$ values of $E_j$ | Higher | Reward units |
| Seed-Level Evaluation Reward | Post-processing | Final $\max(1, \lfloor 0.1 J_s \rfloor)$ values | Higher | Reward units; statistical tests |
| Reward-Threshold Success Proxy | Legacy post-processing | $N^{-1} \sum_{j=1}^{N} S_j$ | Higher | Proportion of checkpoints; not binary success |
| Relative Evaluation-Reward Change | Post-processing | $(E_m - E_b)/\lvert E_b \rvert \times 100\%$ | Higher | Relative percentage, not success rate |
| Collision-Penalty Proxy | Post-processing | $K_c^{-1} \sum_k \lvert c_k \rvert$ | Lower | Mean penalty magnitude |
| Reward Volatility | Post-processing | Rolling sample standard deviation | Lower | Short-term reward variability |
| Convergence Smoothness | Post-processing | Rolling mean of $\lvert \bar{R}_k - \bar{R}_{k-1} \rvert$ | Lower | Smoothed reward change per iteration |
| Late-Stage Reward Std. | Post-processing | Final approximately 10% of $R_k$ | Lower | Late-stage reward variability |
| Coefficient of Variation | Post-processing | $\sigma_R / \lvert \mu_R \rvert$ | Lower | Relative reward variability |
| Convergence Iteration | Post-processing | First iteration reaching $0.9 R_{\text{final}}$ | Lower | Training iterations |
| Training Time | BenchMARL / Post-processing | `timers_total_time` | Lower | Wall-clock time |
| Binary Success Rate | Not available | Episode-level success indicator not logged | Higher | Percentage |
| Direct Collision Rate | Not available | Distinct collision events not logged | Lower | Events per episode or step |

### 3.8.6 Training and Reporting Protocol

In the performance plots, the horizontal axis represents the training iteration index and is labelled *Step*. Let $B$ denote the number of environment frames collected per training iteration. Unless otherwise specified, the reported experiments use

$$B = 30{,}000 \tag{3.38}$$

environment frames per iteration. Experiments using a different collection batch size are identified in their corresponding experimental chapters.

A plotted training iteration $k$ therefore corresponds to approximately

$$F_k = Bk \tag{3.39}$$

collected environment frames. Training statistics are normally recorded at every training iteration, whereas evaluation statistics are recorded only at the configured evaluation intervals.

## 3.9 Implementation Details

All experiments were conducted on a workstation with the following specifications:

- CPU: Intel Core i9-14900KF (3.20 GHz)

- GPU: NVIDIA GeForce RTX 5090 D
- RAM: 128 GB
- Software: Python 3.10; PyTorch 2.8.0.dev20250410+cu128; BenchMARL 1.4.0; VMAS 1.5.0; TorchRL 0.7.2; TensorDict 0.7.2; Optuna 4.3.0; and NumPy 2.2.4. The complete dependency list is provided in `requirements.txt` in the accompanying project archive.

Random seeds are fixed within each experiment and are reported in the corresponding experimental sections and tables. The five-seed Transformer comparison in Chapter 7 uses seeds 26, 38, 48, 66, and 96. The seeds are applied to the Python, NumPy, PyTorch, and CUDA random-number generators. Deterministic CUDA operations are enabled where supported.

# Chapter 4

# Experimental Development and Design Decisions

This chapter analyses the principal implementation challenges encountered during the study and explains the resulting design decisions in relation to previous research. The development observations reported here are used to motivate the controlled experiments in later chapters; unless otherwise stated, they should not be interpreted as independent ablation evidence.

The resulting design decisions informed the controlled experiments presented in the subsequent chapters, which constitute the formal evaluation of the research questions.

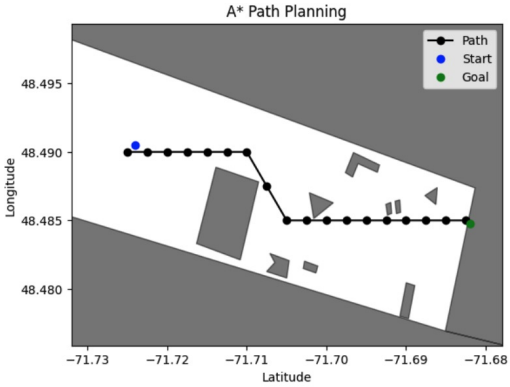
## 4.1 Environment Construction

Environment topology and obstacle distribution can substantially affect the difficulty and comparability of reinforcement-learning navigation tasks. Following the obstacle-dense navigation setting of Lopez-Yepez et al. [4], the environment-development process in this study focused on constructing solvable maps with varied maze structures. The configurations described below represent design-stage evaluations rather than controlled comparisons of map-generation algorithms.

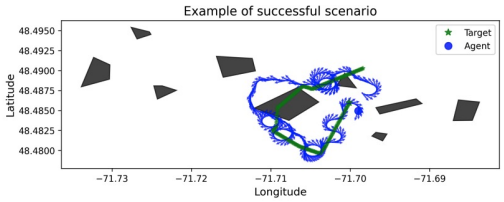
### 4.1.1 Reference Environments from Literature

The initial environment design was based on Lopez et al. (2024) [4]. They provided two example training maps, shown in Figure 4.1.

These environments demonstrated that A* guidance could significantly improve navigation performance. However, the paper did not provide detailed specifications for environment generation, so I developed my own pipeline to address this gap.




(a) Illustration of A*-guided navigation behavior.



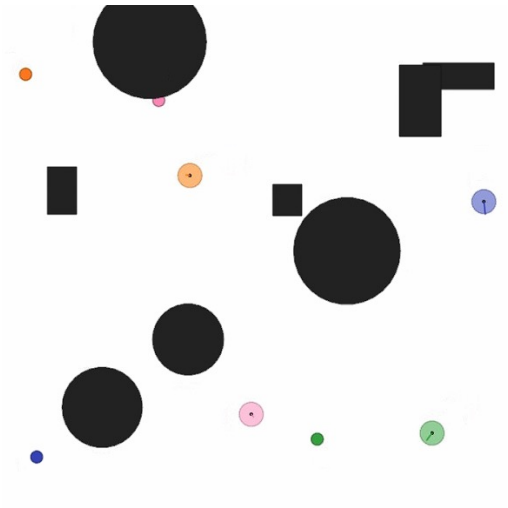

(b) Example navigation environments used in the reference study.

Figure 4.1: Demonstrations from the reference study that informed the design decisions in this thesis[4].

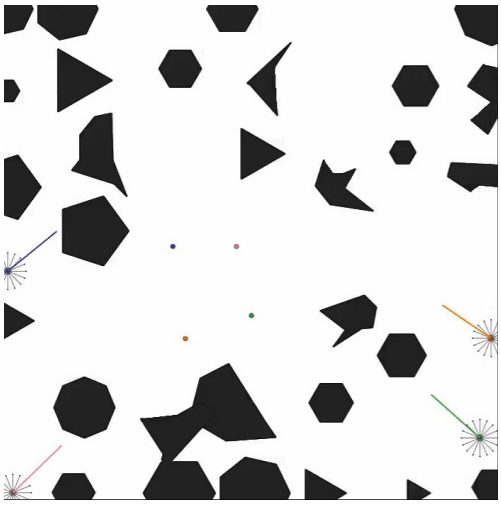
### 4.1.2 First Attempt: High-Density Random Obstacles

I initially created environments with densely packed, randomly placed obstacles. First, I used circles and rectangles, then added random polygons. My goal was to simulate complex real-world scenes with irregularly shaped obstacles.

The generated high-density environments are shown in Figure 4.2.



(a) High-Density Environment with Circular and Rectangular Obstacles of Varying Sizes.



(b) High-Density Environment with Complex Polygonal Obstacles Featuring Concave Structures.

Figure 4.2: High-Density Obstacle Environment Configurations

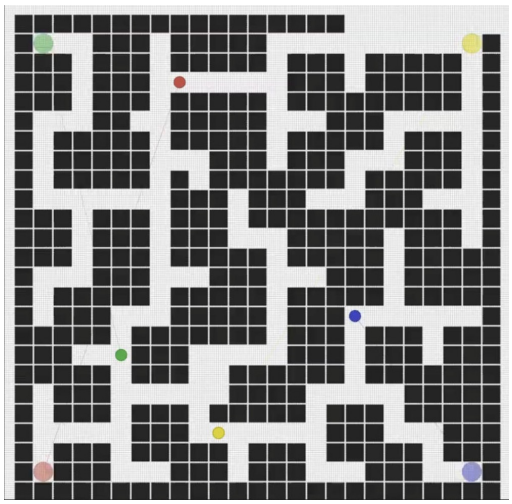
I tested multiple polygon generation strategies: random vertex placement with convex hulls, Voronoi-based decomposition, and changes to obstacle density and size.

While these environments provided visual diversity, they also posed challenges: limited control over complexity, inconsistent start-goal connectivity, and excessive time spent on geometric processing rather than core research. These issues led me to rethink the approach.

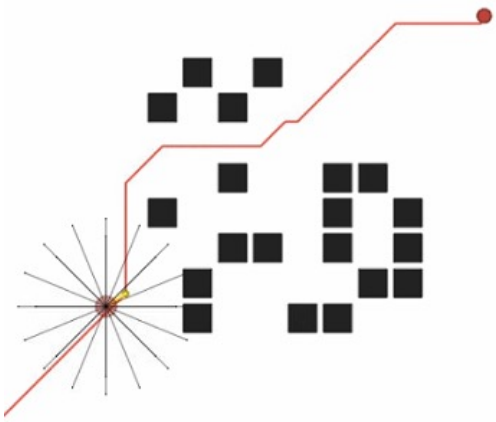
### 4.1.3 Second Attempt: Prim's Algorithm Maze Generation

To improve controllability, I used Prim's algorithm to generate mazes. This ensures a path exists and produces structured layouts.

The generated maze-like environments are shown in Figure 4.3.



(a) High-Density Prim-Generated Maze Environment.



(b) Sparse Prim-Generated Maze Environment.

Figure 4.3: Prim-Based Complex Maze Environment

My first Prim-generated environment was the High-Density Prim-Generated Maze Environment. I expected complex environments to produce stronger agents, but training showed no progress. The agents learned nothing useful. Training logs revealed the problem: the difficulty gap was too large. Starting with complex mazes prevented agents from experiencing success. Without positive feedback, the learning signal was too weak to improve policy.

This experience taught me the value of a gradual shift from simple to difficult environments. Reflecting on this, I generated sparse Prim mazes with fewer corridors and shorter paths to goals (see Figure 4.3).

Training on sparse mazes proceeded more smoothly. Agents began to show learning progress and occasionally reached goals. However, a new problem emerged. I had no clear metrics to define what made a maze "easy" or "hard". The terms "high-density" and "sparse" were subjective descriptions rather than quantifiable measures. Without

proper difficulty metrics, I could not systematically control the training curriculum or fairly compare results across different experiments.

I redesigned the environment pipeline with explicit difficulty criteria to address the issues I had identified.

### 4.1.4 Final Design: Standardized Map Collection

I redesigned the pipeline for reproducibility and control. All maps were $10 \times 10$ units and were generated using fixed Prim algorithm parameters. The complete archive contains 150 maps, with 50 maps in each of the Easy, Medium, and Hard categories. The reported experiments use only the 50 Easy maps: maps 1–30 are used for training, and maps 31–50 are reserved for testing. The Medium and Hard maps are retained for future evaluation.

Figure 4.4 shows examples of maps from each difficulty category.

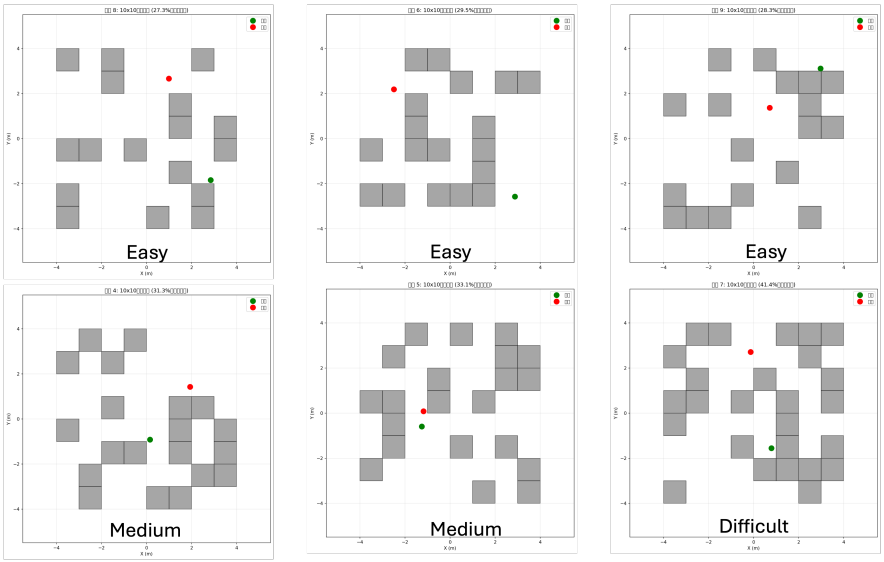



Figure 4.4: Classify maps by difficulty.

The development maps revealed that, without clear boundaries, an agent could bypass obstacles by following the environment edges. Boundary constraints were therefore added so that the path-planning algorithm had to account for the obstacles.

First, the generated maps were enclosed by physical boundary-wall obstacles on all four sides. This prevented an agent from bypassing the internal maze by travelling along or beyond the outer edge of the environment. Second, the continuous $10 \times 10$ environment was discretised into a $50 \times 50$ planning grid, and A* was allowed to expand only nodes whose indices remained within this valid grid.

The obstacle geometries stored in the GeoJSON maps were rasterised into a Boolean

occupancy grid. Cells covered by obstacles were classified as non-traversable and excluded from the set of valid neighbours. Each obstacle region was additionally expanded by a one-cell safety margin. Under the default grid resolution, this corresponds to a nominal clearance of approximately 0.2 environment units and reduces the possibility of generating paths that pass too close to obstacle boundaries.

The planner supports both orthogonal and diagonal movements. For every transition, the destination cell must remain within the map and must not be marked as occupied. For diagonal movements, the two adjacent orthogonal cells are also examined to restrict movement through blocked corners. The feasible neighbour set therefore consists of eight-connected cells that are inside the planning domain, outside the inflated obstacle region, and compliant with the diagonal-transition rule.

These conditions are hard feasibility constraints within the A* planner rather than reward penalties. Collision penalties and LiDAR observations are subsequently used by the reinforcement-learning policy as runtime learning and perception mechanisms; they do not determine whether an A* grid cell is traversable. In addition, the A* path for each agent is computed independently. The planner therefore accounts for the static obstacles represented in the map but does not treat other moving agents as dynamic obstacles.

The final reported experiments use the Easy-map subset described in Chapter 3. Maps from each difficulty level are shown in Figures 3.3a–3.3c.

Table 4.1 summarizes the evolution of environment design throughout this study.

Table 4.1: Evolution of Environment Design

| **Version** | **Method** | **Advantage** | **Limitation** |
|---|---|---|---|
| Reference | Lopez et al. | Validated benchmark | No implementation details |
| Attempt 1 | Random polygons | Visual diversity | Uncontrollable complexity |
| Attempt 2 | Prim maze | Guaranteed paths | No difficulty metrics |
| Final | Standardized collection | Reproducible, labeled | Fixed to single map size |

The standardized environment allowed quantitative comparison and established a unified foundation for algorithm evaluation.

## 4.2 Reward Function Design and A* Integration

This section describes the iterative development of the reward function. The process evolved from simple distance-based rewards to A*-guided shaping mechanisms, with each iteration addressing limitations discovered in the previous version.

Reward shaping is commonly used to provide denser learning signals in tasks with sparse terminal rewards, although poorly aligned shaping terms can produce unintended behaviour [2]. Previous navigation studies have incorporated A*-derived distances,

reference paths, or waypoints into reinforcement learning to provide obstacle-aware guidance [4, 47, 45]. The reward and waypoint decisions examined in this section were motivated by this research direction and adapted to the present VMAS–BenchMARL environment.

### 4.2.1 Early Attempts: Distance-based Rewards

In the early stages, I used a straightforward reward function consisting of three components: goal-distance change, a collision penalty, and a terminal reward for reaching the target. As shown in Figure 4.5, moving closer to the target produces a positive contribution, while moving away results in a penalty.

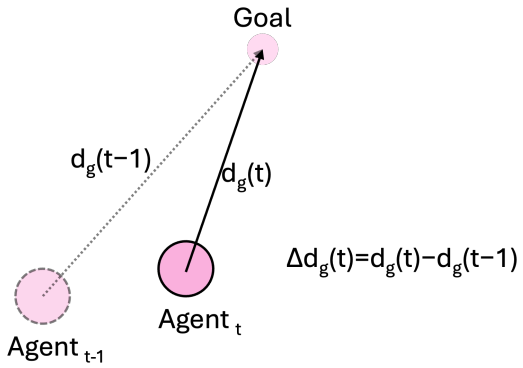



Figure 4.5: Distance-based reward: positive contribution when approaching the target.

This formulation worked reasonably well in simple environments with sparse obstacles. However, it became insufficient in complex maze-like settings. The main problem was sparse feedback: agents received meaningful rewards only when they happened to move in the correct direction. In maze environments, the goal is often not directly reachable, and the Euclidean distance can be misleading.

### 4.2.2 Introducing A* Path Guidance

To address the sparse reward problem, I incorporated A* pathfinding into the reward design. The idea was inspired by Lopez et al. (2024)[4]: use the optimal path computed by A* as a reference for reward shaping.

The integration works as follows:

1. A* computes the shortest path from agent position to goal
2. The path is decomposed into a sequence of waypoints
3. The agent receives rewards based on progress toward the next waypoint

This design provides continuous directional feedback even when the goal is not directly visible. The agent learns to follow the planned path rather than blindly moving toward the Euclidean goal direction.

Since the original implementation was not publicly available, I reimplemented the waypoint-based mechanism based on the paper's descriptions and my own experimental observations. The technical details are presented in Chapter 3.

### 4.2.3 Waypoint Selection: From Oscillation to Stability

The initial implementation simply selected the nearest point on the A* path. Ideally, the agent should reach each waypoint sequentially before moving forward, but as shown in Figure 4.6, the reality is different.

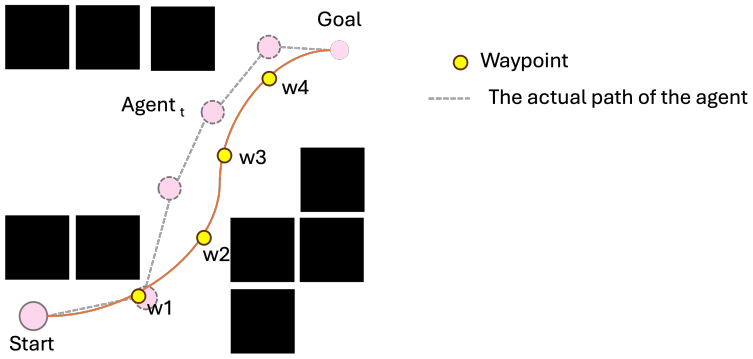



Figure 4.6: The actual agent path does not strictly follow the waypoint.

During preliminary development, nearest-node selection was observed to produce frequent changes in the guidance target in some trajectories. This coincided with oscillatory behaviour and inconsistent forward motion in the illustrated example. Figure 4.7 presents this qualitative development observation; it was not obtained from a controlled comparison.

Based on this qualitative development observation, I implemented a look-ahead waypoint selection mechanism. Instead of targeting the nearest path node, the mechanism selects the first subsequent waypoint whose distance from the agent is at least $d_{\min}$. The experiments used a fixed value of $d_{\min} = 0.5$, but this value was not evaluated through a systematic parameter search or controlled ablation study.

In one development run, the introduction of this rule coincided with a visually smoother trajectory and more consistent forward motion. Figure 4.9 presents an illustrative example from that run. However, the look-ahead rule was not independently compared with nearest-node selection under controlled multi-seed experiments. The example therefore provides qualitative development evidence only and does not establish an improvement in trajectory smoothness, convergence, or task performance.

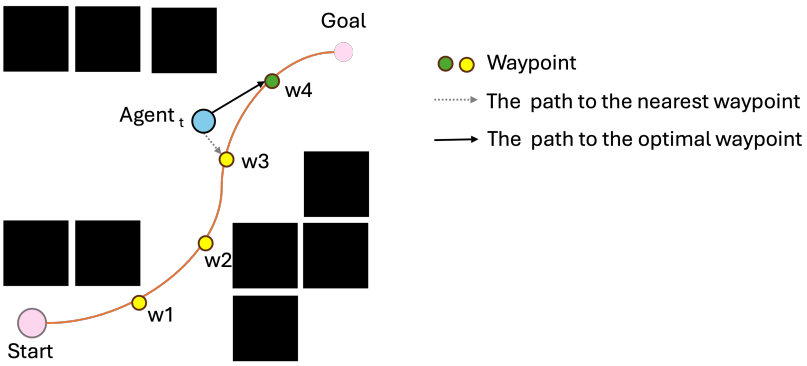



Figure 4.7: The nearest waypoint is not the optimal waypoint. The agent (blue) is closest to the waypoint (w3), but the optimal connection is the waypoint (green).

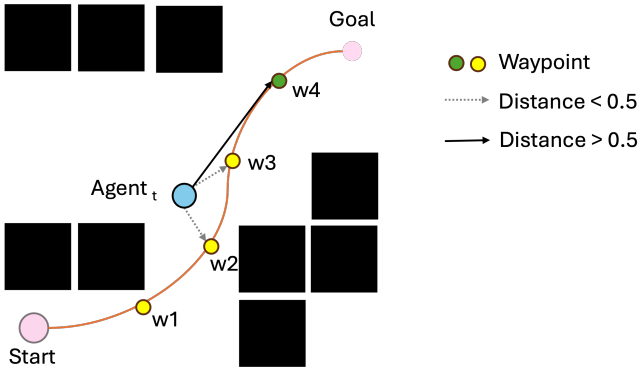



Figure 4.8: Waypoint selection based on a distance threshold. After identifying the nearest path node, the mechanism scans the subsequent nodes and selects the first waypoint whose distance from the agent is at least the fixed threshold $d_{\min} = 0.5$. In this illustrative example, $w4$ is selected as the active waypoint.

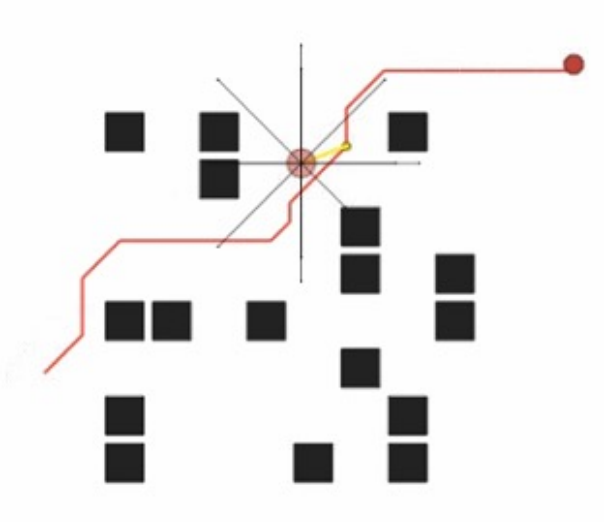


Figure 4.9: Example of successful pathfinding for an agent using A*-guided rewards.

### 4.2.4 Remaining Challenge: The Trap Problem

Dead ends, insufficient exploration, and locally misleading reward signals are recognised difficulties in reinforcement-learning-based navigation. Additional penalties may discourage stagnation or repeated behaviour, but complex reward structures can also introduce unintended incentives if they are not systematically validated [2].

Despite these improvements, a critical problem remained: agents frequently got stuck in dead ends. Figure 4.10 illustrates this trap phenomenon.

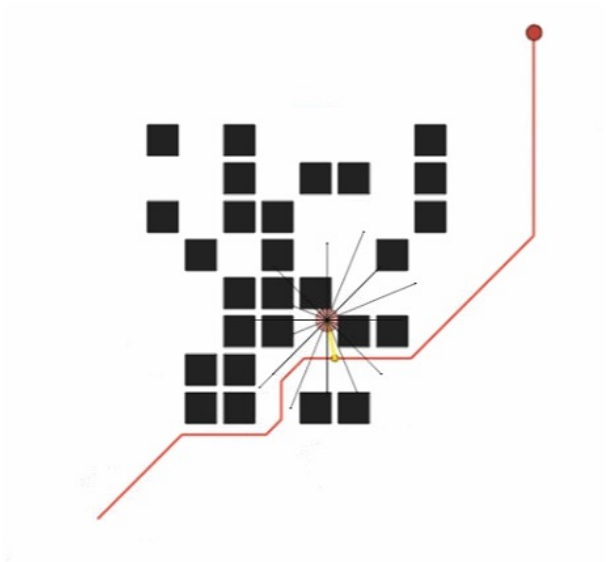


Figure 4.10: Example of an agent trapped in a dead end despite A* guidance.

The trap problem occurs because A* provides path-level guidance but cannot help the agent escape once it enters a dead end. The A* path assumes the agent can execute precise movements, but learned policies have execution errors that accumulate over time.

Based on these recognised failure modes, the following additional reward terms were examined:

- Stagnation penalty: Negative reward for staying in the same position
- Rotation penalty: Negative reward for repeated turning without forward progress
- Visited-state penalty: Negative reward for returning to previously visited locations

However, these modifications did not solve the problem. I adjusted the parameters too aggressively without systematic validation, leading to training instability. The penalties sometimes discouraged necessary exploration or caused agents to avoid valid paths.

This experience revealed a fundamental limitation: reward shaping alone cannot solve problems that require temporal reasoning. To escape a trap, an agent must recognize that it has been stuck and recall which directions have already failed. This requires memory of past states, which a simple feedforward policy cannot provide.

This realization motivated the exploration of memory-enhanced architectures, but before that, I'd like to mention some other optimizations I've made.

## 4.3 Performance Optimization Attempts

To improve training stability and overall performance, several optimization strategies were explored, including automated hyperparameter tuning, model and data scaling, and alternative MARL algorithms.

Reinforcement-learning performance is sensitive to hyperparameter selection, model capacity, algorithm choice, and random initialization. The optimization attempts in this section were therefore used to identify workable experimental configurations rather than to establish the independent effectiveness of individual optimization techniques.

### 4.3.1 Automated Hyperparameter Optimization

Optuna was integrated into the experimental pipeline to automate hyperparameter search in the initial phase of the study. It was observed that Optuna could produce parameter configurations associated with improved training stability.

Optuna was primarily employed in the early stage of this study to establish baseline configurations. Despite its effectiveness, the practical cost of hyperparameter optimization became prohibitive as the experiments progressed. A single trial often required up to one day of training, and completing a full optimization study typically spanned several days. Moreover, training instability led to frequent premature termination of trials. For these reasons, Optuna was not adopted in the later stages of this research.

### 4.3.2 Model Scaling Exploration

Following the recommendations of Lopez et al. (2024) [4], further attempts were made to improve performance through scaling. These included expanding the training dataset to include more diverse maze configurations and increasing network capacity by experimenting with deeper, wider architectures.

However, these modifications did not substantially alleviate the trap problem. Neither convergence speed nor training stability improved consistently. This observation suggests that scaling model capacity or training data alone is insufficient to address the task's underlying challenges.

As a result, the study shifted to exploring alternative multi-agent reinforcement learning algorithms.

### 4.3.3 Algorithm Exploration

The examined algorithms represent different MARL learning paradigms, including policy-gradient, actor–critic, and value-factorization approaches [16, 15, 14].

To understand whether the trap problem was caused by poor parameter tuning or by the learning algorithm itself, I tested alternative multi-agent reinforcement learning algorithms in addition to MAPPO. In particular, I experimented with QMIX and MADDPG, which learns policies in a different way from MAPPO. The goal of this exploration was not to achieve the best possible performance, but to examine whether changing the learning mechanism could help agents escape from traps in complex maze environments.

Switching to other algorithms did not produce a clear descriptive improvement in training stability or convergence in these development runs. The agents still frequently became stuck on suboptimal paths, especially in complex mazes. This observation suggests that the trap problem may involve structural navigation challenges rather than only parameter tuning or algorithm selection, but the underlying cause was not separately tested.

## 4.4 Architecture Exploration for Temporal and Structured Representation

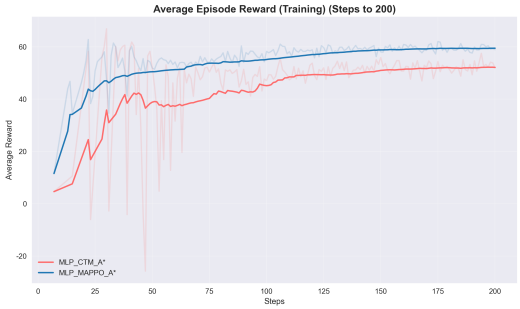
Architecture exploration was guided by prior work on iterative computation, recurrent sequence modelling, and attention-based representation learning. CTM was examined for iterative internal processing [50], LSTM for recurrent temporal representation [51], and the Transformer for attention-based parallel processing [1]. These development runs informed the selection of architectures for later controlled experiments but were not designed as a complete architecture ablation.

The trap problem remained difficult to resolve through reward shaping or network scaling. Under my supervisor's guidance, I reconsidered whether the model architecture lacked the capacity for long-term dependencies.

Through repeated experiments, I realized that agents often fail in traps because they lack sufficient temporal context. They cannot recognize that they have entered a state of repetition or stagnation. This motivated exploration of memory-enhanced models.

### 4.4.1 Continuous Thought Machines (CTM)

I first implemented the Continuous Thought Machine (CTM)[50]. CTM introduces an explicit internal time dimension into neural computation. It enables the model to process through multiple internal steps rather than a single forward pass. After implementing the Continuous Thought Machine (CTM), we observed that its performance on the MARL tasks did not meet expectations. The result is illustrated in Figure 4.11.




(a) Average Episode Reward


(b) Evaluation Reward (legacy embedded plot label: "Success Rate")

Figure 4.11: CTM Performance Deviating from Expected Behavior in MARL Tasks.

To address this limitation, my supervisor suggested exploring pre-trained models in order to incorporate stronger prior knowledge. However, further investigation revealed a fundamental incompatibility between the data representations required by BenchMARL and the input interfaces expected by existing pre-trained models. Adapting either side would require substantial modifications to the training pipeline and model architecture, going beyond the intended scope of this work. Consequently, the pre-training direction was not pursued further.

### 4.4.2 Long Short-Term Memory Networks (LSTM)

I then transitioned to an LSTM-based architecture [51]. Despite its theoretical suitability, LSTM exhibited extremely long training times in practice. The recurrent nature enforces strict sequential computation. Each step depends on the previous one. This prevents parallelization, significantly increasing training time.

As a result, full-scale experiments became computationally prohibitive. Convergence proceeded too slowly to be practical.

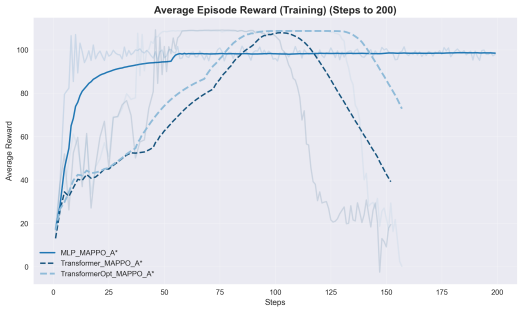
### 4.4.3 Transformer-Based Architecture

To address the training efficiency issue, I replaced the LSTM with a Transformer-based model [1]. Transformers remove recurrent computation and use self-attention mechanisms, enabling efficient parallel processing across input tokens. In this study,

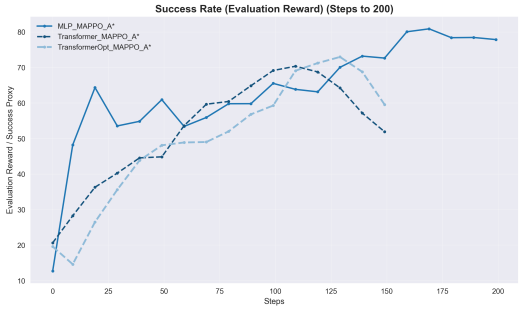
the Transformer receives only current-timestep observation tokens. It therefore models relationships within the current observation but does not capture long-range temporal structure or provide memory of previous states.

In this study, only the encoder component is utilized. Since the task does not require autoregressive generation, the decoder is unnecessary. This reduces computational cost and simplifies the architecture.

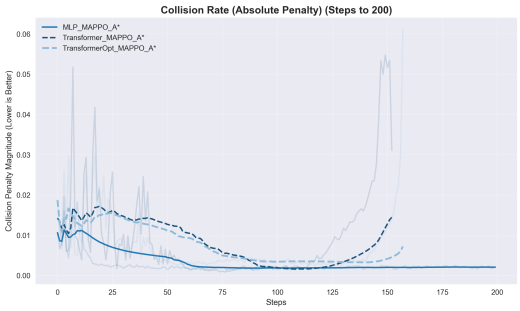
To control for experimental variables, we initially enforced the Transformer and the MLP to share the same parameter budget. In addition, several training hyperparameters—including the learning rate, entropy coefficient, PPO clipping range, and the number of minibatch iterations—were harmonized across both architectures. Contrary to expectations, this strategy led to chaotic and unstable training dynamics, as illustrated in Figure 4.12.



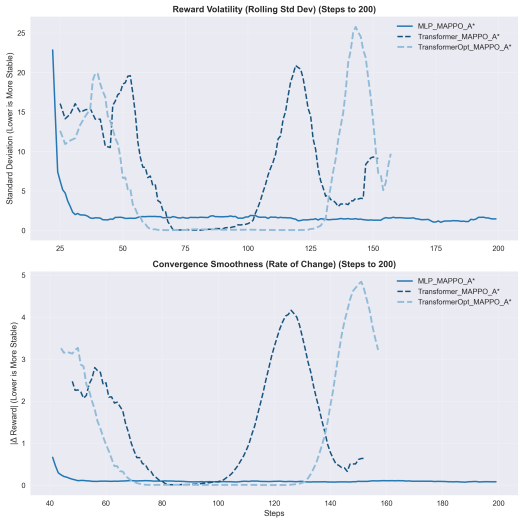

(a) Average Episode Reward




(b) Evaluation Reward (legacy embedded plot label: "Success Rate")




(c) Collision-Penalty Proxy




(d) Convergence Stability

Figure 4.12: Failure Case of Parameter-Aligned Training between Transformer and MLP in Single-Agent Navigation.

A deeper analysis revealed that these hyperparameters interact fundamentally differently with Transformer-based policies compared to MLPs. Owing to attention mechanisms and token-level representations, Transformers exhibit higher gradient variance and increased sensitivity to optimization settings. As a result, hyperparameter values that are stable for MLPs can severely disrupt the training dynamics of Transformers. This observation suggests that enforcing uniform training configurations across heterogeneous architectures does not ensure fairness; instead, it suppresses the inductive advantages of the Transformer and leads to misleading performance degradation.

Even after relaxing the parameter alignment constraints, the Transformer exhibited clear signs of overfitting during the early stages of training. An attempt was made to alleviate this issue by reducing model size and tuning architectural parameters. However, this intervention resulted in a noticeable decline in overall performance and poor generalization, indicating that overfitting was not solely attributable to model capacity.

Motivated by this observation, we revisited the related literature and explored a complete waypoint-aware Transformer configuration that incorporated A*-derived waypoint observations. After this configuration change, the evaluation reward showed a positive descriptive difference. Because its individual components were not tested separately in these development runs, the difference cannot be attributed specifically to positional encoding, gated feed-forward layers, or local attention. The observation nevertheless motivated the controlled Standard Transformer versus TA-WAT comparison reported later.

## 4.5 Lessons Learned

Throughout this research journey, several key insights emerged that may benefit future work in similar domains.

### 4.5.1 Curriculum matters more than complexity

My first attempt at training directly on high-density mazes resulted in complete failure. Agents received no positive reward signals and learned nothing. Only after switching to sparse mazes did training begin to progress. This taught me that agents need early success experiences to bootstrap learning. Complex environments should be introduced gradually, not from the start.

### 4.5.2 Fair comparison requires architecture-specific tuning

When I forced the Transformer and MLP to share identical hyperparameters, the Transformer training became chaotic while the MLP trained normally. I initially interpreted this as meaning that Transformer was unsuitable for the task. Later analysis revealed that Transformers have fundamentally different sensitivity to learning rates and entropy coefficients. True fairness means allowing each architecture to operate at its best, not handicapping one with settings optimized for another.

### 4.5.3 Automation has hidden costs

Optuna consistently found better hyperparameters than manual search. However, a single trial could take up to a day, and training instability led many trials to fail early. The total time required for proper optimization exceeded what was practical for this project. This forced me to rely on manual tuning informed by intuition rather than systematic search.

### 4.5.4 Missing details require substantial reimplementation effort.

The reference paper by Lopez et al. demonstrated promising results but did not describe how to generate the training environments. I had to develop the entire environment construction pipeline independently, including map generation, difficulty classification, and standardization procedures. This consumed significant time that could have been spent on algorithm development.

### 4.5.5 Failed attempts inform successful solutions

Many approaches explored in this work did not produce useful descriptive results: CTM performed poorly, LSTM was too slow, model scaling did not help, and alternative algorithms showed no clear improvement. However, each outcome narrowed the hypothesis space. The later waypoint-aware configuration showed a positive descriptive difference in selected metrics, motivating the controlled architectural comparison in Chapter 7. These development observations do not establish the effect of any individual Transformer component.

# Chapter 5

# Baseline Reproduction and Validation

## 5.1 Introduction

Path planning is a fundamental problem in robotics. In this task, agents—autonomous entities such as robots—must navigate from a start position to a goal while avoiding obstacles. Traditional methods such as A* provide optimal solutions, but they require complete environmental knowledge and struggle with dynamic, changing scenarios. Deep Reinforcement Learning (DRL), an area of machine learning in which agents learn through trial and error by receiving rewards or penalties, has shown promise for learning adaptive navigation policies. However, DRL often suffers from poor sample efficiency—the amount of data an agent needs to learn effectively—and slow convergence, especially in environments dense with obstacles.

Lopez-Yepez et al. [4] proposed integrating A* pathfinding with DRL to address these limitations. Their key insight is to use A*-generated waypoints as intrinsic reward signals during training. This approach guides exploration without needing map information at execution time. Their experiments demonstrated significant improvements in single-agent scenarios. However, the generalizability to multi-agent settings remains unexplored.

This chapter reproduces and validates their core findings within the VMAS (Virtual Multi-Agent Simulator) framework. Although VMAS is designed for multi-agent reinforcement learning—a method in which multiple agents learn and make decisions simultaneously—we intentionally configure single-agent scenarios to enable a fair baseline comparison with the original study. This approach isolates the effects of A* integration (a heuristic search algorithm) from the complexities of multi-agent coordination and lays the groundwork for experiments in later chapters.

## 5.2 Experimental Design

### 5.2.1 Research Hypotheses

Based on the findings of Lopez-Yepez et al., we formulate three hypotheses:

- H1: Incorporating A* demonstration trajectories improves sample efficiency and convergence speed of MAPPO.
- H2: A* guidance provides stronger benefits in compact grid-world environments where exploration is spatially constrained.
- H3: A* initialization leads to more stable early-stage learning compared to training MAPPO from scratch.

### 5.2.2 Experimental Setup

The experiments are conducted in VMAS (Vectorized Multi-Agent Simulator), a GPU-accelerated 2D physics engine integrated with BenchMARL—a benchmarking suite for Multi-Agent Reinforcement Learning—to enable standardized algorithm implementation. The key configurations are summarized in Table 5.1.

Table 5.1: Experiment Configuration Summary

| **Parameter** | **Value** | **Parameter** | **Value** |
|---|---|---|---|
| World Size | $10 \times 10$ units | Training Frames | 30,000,000 |
| Grid Resolution | $50 \times 50$ cells | Parallel Environments | 120 |
| Agent Radius | 0.25 units | Batch Size | 60,000 frames |
| LiDAR Rays | 16 (360°) | Learning Rate | $5 \times 10^{-4}$ |
| Max Episode Steps | 500 | Random Seed | 42 |
| Map Difficulty | Easy (50 maps) | Evaluation Episodes | 200 |

The observation space has 26 dimensions when A* is enabled, and 22 without it. It includes agent state (its position and velocity), the position of the goal relative to the agent, LiDAR readings, and, optionally, A* waypoint information. This design follows the mapless navigation paradigm introduced by Lopez-Yepez et al. In this paradigm, the agent relies on local sensing rather than global map knowledge during execution.

### 5.2.3 Baseline Comparison

Two configurations are compared:

1. **MLP-MAPPO (without A*):** Standard MAPPO with distance-based reward shaping only. Serves as the performance baseline.

2. **MLP-MAPPO (with A*):** MAPPO is augmented with A*-generated waypoints as additional reward guidance signals. The A* algorithm computes optimal paths during training to provide shaped rewards, but this information is **not** available during evaluation.

During the training phase, the reward function follows the formulation in Equation 5.1:

$$r_t = G(t) - C(t) - \lambda_d \cdot \Delta d_{\text{goal}} - \lambda_{A^*} \cdot \Delta d_{\text{waypoint}} \tag{5.1}$$

where $G(t) = 10.0$ for goal achievement, $C(t) = 0.05$ for collision penalty, $\lambda_d = 1.0$, and $\lambda_{A^*} = 2.0$.

During the evaluating phase, the reward function excludes the A*-guided waypoint term. The evaluation reward function is defined in Equation 5.2:

$$r_t = G(t) - C(t) - \lambda_d \cdot \Delta d_{\text{goal}} \tag{5.2}$$

where $G(t) = 10.0$ denotes the reward for reaching the goal, $C(t) = 0.05$ represents the collision penalty, and $\lambda_d = 1.0$.

## 5.3 Results

### 5.3.1 Overall Performance Comparison

To highlight performance differences, Table 5.2 presents a comparison between the two configurations.

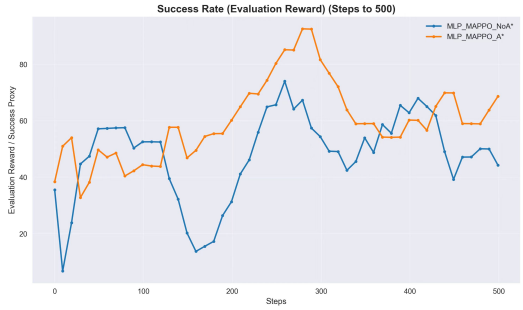
Table 5.2: Performance Comparison: MLP-MAPPO with and without A* Guidance. Higher values indicate better performance for Avg Reward and Final-Window Evaluation Reward, while lower values indicate better performance for the Collision-Penalty Proxy and Training Time. In the Change row, positive values indicate an increase relative to the baseline (Without A*), while negative values indicate a decrease.

| Configuration | Avg Reward | Final-Window Evaluation Reward | Collision-Penalty Proxy | Training Time |
|---|---|---|---|---|
| Without A* | 106.30 | 44.30 | 0.0050 | 5h 21m |
| With A* | 107.44 | 68.64 | 0.0030 | 18h 9m |
| **Change (** | +1.1% | +54.9% (better) | -40.0%(better) | +239%(worse) |

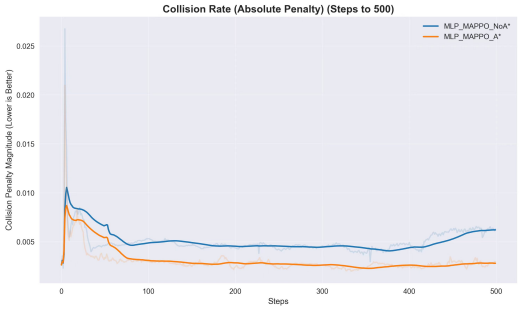
The results reveal a nuanced trade-off: the A*-guided run has a 54.9% higher final-window evaluation-reward value and a 40.0% lower collision-penalty proxy, but requires 239% more training time. This finding is consistent with the previous observations by Lopez-Yepez et al., who noted that A* computation adds non-negligible overhead during training. These are descriptive reward and proxy differences rather than direct success or collision rates.

### 5.3.2 Learning Dynamics Analysis

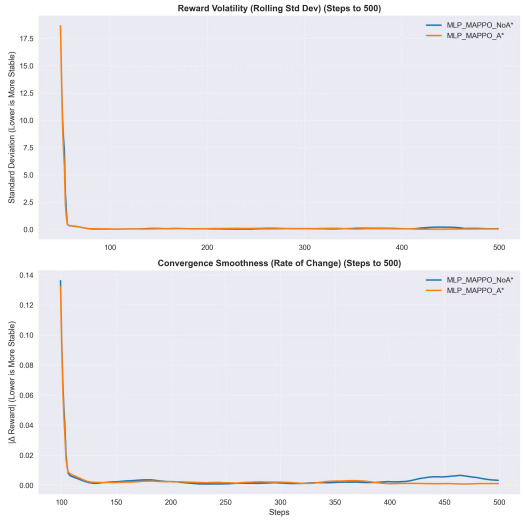

(a) Average Episode Reward




(b) Evaluation Reward (legacy embedded plot label: "Success Rate")




(c) Collision-Penalty Proxy




(d) Convergence Stability

Figure 5.1: Training dynamics comparison for single-agent navigation (Steps 0–500).

Figure 5.1 illustrates the training and evaluation behavior of MLP-based MAPPO with and without A* guidance over 500 training iterations. The average episode reward curves (a) show that both variants improve during the early training phase and reach stable ranges after approximately 100 training iterations. Their observed convergence speeds are similar, while MLP-MAPPO with A* reaches a slightly higher reward plateau in this run. This is a descriptive difference and was not subjected to a significance test.

The auxiliary metrics show further descriptive differences. The collision-penalty proxy curves (c) show that MLP-MAPPO with A* has a faster reduction in the mean magnitude of the logged collision-penalty statistic and maintains a lower steady-state value. The reward volatility and convergence smoothness plots (d) show that both methods stabilize quickly after convergence, but the A* variant exhibits marginally lower fluctuations and smoother reward changes. The evaluation-reward curve (b) is

also generally higher for MLP-MAPPO with A* across evaluation checkpoints. Evaluation reward is not a binary success rate, and the collision-penalty proxy is not a direct collision frequency.

## 5.4 Hypothesis Validation and Analysis

### 5.4.1 H1: Sample Efficiency and Convergence Speed

Table 5.3 reports the sample-efficiency and convergence measures used to evaluate H1.

Table 5.3: H1 Validation: Sample Efficiency Metrics

| Metric | Without A* | With A* | Change | Conclusion |
|---|---|---|---|---|
| Total Training Frames | 31M | 31M | 0% | Not Supported |
| Iteration-Normalized Reward Cost | 5.00 | 4.99 | -0.2% | Marginally Supported |
| Training Time | 5.36h | 18.15h | +239%(worse) | Not Supported |

**Result: Partially Supported.**

The A* group has a marginally lower iteration-normalized reward cost but does not reduce the total number of training frames. The substantial increase in training time is due to A* path computation overhead. This limitation is acknowledged by Lopez-Yepez et al., who noted that the computation of the A* heuristic can take a non-negligible amount of time during training.

**Analysis:** Unlike the original study, which used a simpler single-agent setup, our VMAS-based implementation may introduce extra computational overhead from vectorized environment management. The lack of convergence acceleration may be due to our relatively large batch size (60,000 frames per batch). This setting differs substantially from typical single-agent DRL setups and may dilute the immediate feedback benefits of A* waypoint guidance.

### 5.4.2 H2: Benefits in Compact Environments

Table 5.4 reports the reward, collision-penalty proxy, and final-window evaluation reward used to evaluate H2.

Table 5.4: H2 Validation: Environment Constraint Effects

| Metric | Without A* | With A* | Change | Conclusion |
|---|---|---|---|---|
| Average Reward | 106.30 | 107.44 | +1.1% | Supported |
| Collision-Penalty Proxy | 0.0050 | 0.0030 | -40.0% (better) | Strongly Supported |
| Final-Window Evaluation Reward | 44.30 | 68.64 | +54.9% (better) | Strongly Supported |

**Result: Supported.**

The A* guidance produces positive descriptive differences in the compact 10×10 grid-world, including a 40.0% lower collision-penalty proxy and a 54.9% higher final-window evaluation-reward value. These reward-based measures are consistent with more favourable evaluation outcomes when exploration is spatially constrained, but they do not directly establish collision avoidance or binary goal-reaching success.

**Analysis:** The higher final-window evaluation reward is consistent with more favourable evaluation outcomes in the A*-guided run. The 40.0% lower collision-penalty proxy is likewise consistent with a smaller mean magnitude of the logged collision-penalty statistic. Because neither measure is a direct event count, these results do not establish binary task success or collision frequency.

### 5.4.3 H3: Early-Stage Learning Stability

Table 5.5 reports the stability measures used to evaluate H3.

Table 5.5: H3 Validation: Stability Metrics

| Metric | Without A* | With A* | Change | Conclusion |
|---|---|---|---|---|
| Coefficient of Variation | 0.058 | 0.058 | +0.0% | Similar |
| Reward Std. Dev. | 6.21 | 6.20 | -0.2% | Similar |
| Late-Stage Reward Std. Dev. | 0.06 | 0.04 | -33.3% (better) | Supported |

**Result: Partially Supported.**

While overall variance metrics are similar, the late-stage reward standard deviation shows a 33.3% reduction with A* guidance, indicating lower reward variability in the final portion of training. However, the effect is not consistent across all stability measures.

**Analysis:** The partial support for H3 suggests that A* guidance provides localized stability improvements rather than global variance reduction. This may be because the fundamental MAPPO optimization dynamics remain unchanged. A* affects the reward landscape but not the policy gradient mechanics. The improved rolling stability likely reflects more consistent reward signals from waypoint guidance during intermediate navigation phases.

## 5.5 Discussion

Our results partially reproduce the findings of Lopez-Yepez et al., with notable similarities and differences:

**Consistent Findings:**

- The A*-guided run has a higher final-window evaluation reward

- The collision-penalty proxy is lower with waypoint guidance
- Computational overhead from A* path computation is substantial

**Divergent Observations:**

- Sample efficiency gains are marginal in our setup (1.1% vs. their reported substantial improvements)
- Convergence speed is not improved despite better final performance

## 5.6 Summary

This experiment reproduces the baseline study by Lopez-Yepez et al. and evaluates A* integration with DRL for single-agent navigation. Table 5.6 summarizes the outcome for each hypothesis.

Table 5.6: Hypothesis Validation Summary

| Hypothesis | Core Statement | Result |
|---|---|---|
| H1 | A* improves sample efficiency and convergence speed | Partially Supported |
| H2 | A* provides stronger benefits in compact environments | **Supported** |
| H3 | A* stabilizes early-stage learning | Partially Supported |

**Key Takeaways:**

1. The A*-guided run has a 54.9% higher final-window evaluation-reward value and a 40.0% lower collision-penalty proxy. These are descriptive reward and proxy differences rather than direct success or collision rates, and they do not confirm a general performance effect.
2. The computational overhead (+239% training time) represents a substantial trade-off that must be considered in practical applications.
3. The largest observed differences occur in final-window evaluation reward and the collision-penalty proxy rather than in convergence speed. This pattern is descriptive and does not establish the mechanism by which A* guidance affects learning.

These findings justify extending other MARL Algorithms, which is explored in Experiment 2.

# Chapter 6

# Cross-Algorithm Evaluation of A*-Guided Learning

## 6.1 Introduction

Chapter 5 reported descriptive differences between MAPPO runs with and without A* guidance. Those observations are compared with the findings of Lopez-Yepez et al. [4], but they do not independently confirm a general improvement. Although multi-agent algorithms are used, the task remains single-agent to ensure controlled comparisons.

However, Lopez-Yepez et al. noted in their future work section that "exploring the application of the proposed method in scenarios involving multiple autonomous agents" remains an open research direction. Before extending to multi-agent scenarios, a fundamental question must be addressed: Does the A* integration approach generalize across different multi-agent reinforcement learning algorithms?

This experiment investigates the generalization of A* integration by evaluating three representative multi-agent reinforcement learning (MARL) algorithms: MAPPO, MADDPG, and QMIX. All algorithms are tested under identical conditions. Following the methodology from Chapter 5, we systematically compare each algorithm's response to A* guidance, allowing us to identify which algorithmic paradigms are most compatible with path planning integration.

## 6.2 Experimental Design

### 6.2.1 Research Hypotheses

- H1: A* guidance consistently reduces collisions in single-agent scenarios across MARL algorithms, as waypoint information resolves credit assignment regardless

of the learning paradigm.

- H2: Continuous-action algorithms (MAPPO, MADDPG) more precisely follow A* waypoints than discrete-action algorithms (QMIX), yielding greater performance.
- H3: Experience replay in off-policy algorithms amplifies the impact of A*-guided trajectories.

### 6.2.2 Experimental Setup

The main training hyperparameters follow the settings used in previous experiments. However, to reduce training time due to many experiments, some hyperparameters related to efficiency, such as batch size, are slightly adjusted. These changes aim to accelerate training rather than boost performance.

Table 6.1 summarizes the experimental configuration used in this chapter.

Table 6.1: Experiment Configuration Summary

| **Parameter** | **Value** | **Parameter** | **Value** |
|---|---|---|---|
| World Size | $10 \times 10$ units | Training Frames | **6,000,000** |
| Grid Resolution | $50 \times 50$ cells | Parallel Environments | 120 |
| Agent Radius | **0.15** units | Batch Size | **30,000 frames** |
| LiDAR Rays | 16 (360°) | Learning Rate | $5 \times 10^{-4}$ |
| Max Episode Steps | 500 | Random Seed | **38** |
| Map Difficulty | Easy (50 maps) | Evaluation Episodes | **100** |

### 6.2.3 Algorithm Selection

Three algorithms representing distinct learning paradigms are evaluated. Table 6.2 summarizes their main characteristics.

Table 6.2: Algorithm Characteristics

| **Property** | **MAPPO** | **MADDPG** | **QMIX** |
|---|---|---|---|
| Learning Type | On-Policy | Off-Policy | Off-Policy |
| Action Space | Continuous | Continuous | Discrete |
| Policy Type | Stochastic | Deterministic | Value-Based |

### 6.2.4 Baselines for Comparison

A total of six experiments are conducted in this study. For each algorithm, two training configurations are considered: one with A* guidance during training and one without A*. During evaluation, A* is not used for any model.

1. MAPPO-A*: MAPPO with A* waypoint guidance
2. MAPPO-Base: MAPPO without A* guidance
3. MADDPG-A*: MADDPG with A* waypoint guidance
4. MADDPG-Base: MADDPG without A* guidance
5. QMIX-A*: QMIX with A* waypoint guidance
6. QMIX-Base: QMIX without A* guidance

## 6.3 Results

### 6.3.1 Overall Performance Comparison

Table 6.3: Performance Comparison of Different Algorithms

| Configuration | Avg Reward | Final-Window Evaluation Reward | Collision-Penalty Proxy | Convergence Iteration | Training Time |
|---|---|---|---|---|---|
| MADDPG-A* | **106.94** | **80.68** | 0.0049 | 22 | 3h 8m |
| MADDPG-Base | 105.83 | 75.70 | 0.0056 | 24 | 2h 10m |
| MAPPO-A* | 97.45 | 77.87 | **0.0026** | **18** | 2h 22m |
| MAPPO-Base | 106.91 | 79.14 | 0.0032 | **18** | **1h 12m** |
| QMIX-A* | 100.63 | 68.19 | 0.0047 | 59 | 2h 47m |
| QMIX-Base | 100.79 | 68.74 | 0.0060 | 65 | 2h 6m |

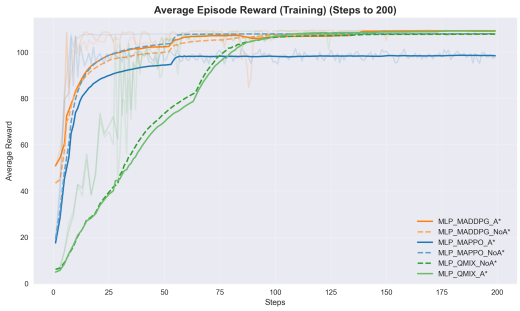
When comparing the average episode reward between Experiment 1 and Experiment 2, the primary difference lies in the batch size configuration. In Experiment 1, MAPPO with A* achieves higher average episode reward than its counterpart without A*. However, this trend is reversed in Experiment 2.

The main difference between the two experiments is a reduction in batch size from 60,000 to 30,000. This suggests that MAPPO is particularly sensitive to changes in batch size. MAPPO, as an on-policy algorithm, relies on stable gradient estimates, which are improved by larger batches.

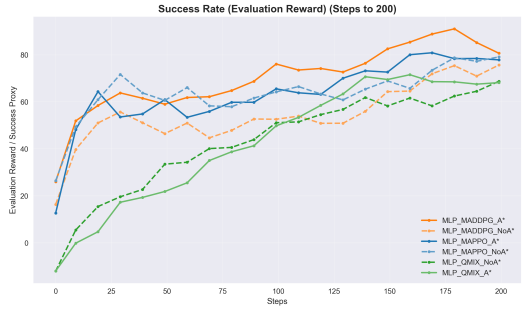
A* guidance produces more structured and less diverse trajectories. With a smaller batch size, reduced diversity may lead to higher gradient variance and less stable updates. This can diminish the performance gains observed with A* guidance.

Results in Table 6.3 show that A* guidance does not increase the final-window evaluation reward for all algorithms. The reported values are averaged over the final $\min(10, J)$ valid evaluation observations, as defined in Equation 3.11.

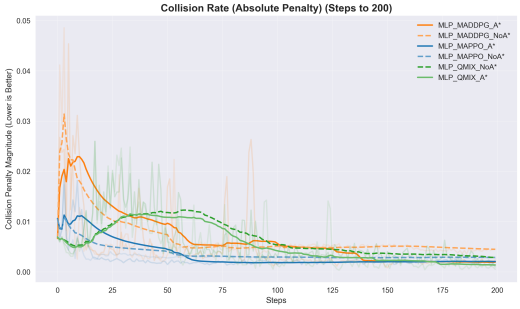
As shown in Figure 6.1, MAPPO and QMIX with A* display a delayed descriptive difference. Their evaluation-reward values surpass those of the baseline after approximately training iteration 100.



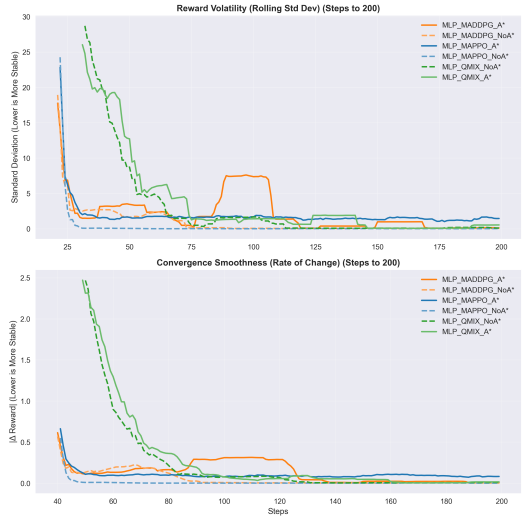

(a) Average Episode Reward




(b) Evaluation Reward (legacy embedded plot label: "Success Rate")




(c) Collision-Penalty Proxy




(d) Convergence Stability

Figure 6.1: Training dynamics comparison of different algorithms for single-agent navigation, evaluated without A* guidance.

### 6.3.2 Learning Dynamics Analysis

Figure 6.1 shows training progress over 200 training iterations.

Figure 6.1 illustrates the training progression over 200 training iterations. The average reward curves (a) reveal distinct algorithm characteristics: MADDPG converges fastest, reaching stable performance within 25 iterations; MAPPO shows rapid initial learning but plateaus at different levels depending on A* configuration; QMIX exhibits the slowest convergence with high variance during middle training stages. Notably, MAPPO-A* achieves lower average reward than its baseline, suggesting sensitivity to batch size when combined with structured guidance. The evaluation-reward curves (b) show that MADDPG-A* has the highest final value (approximately 80 reward units), while MAPPO configurations show comparable values and QMIX maintains the lowest values across the tested conditions.

The collision-penalty proxy analysis (c) shows lower proxy values for the A*-guided single-agent runs across all three algorithms. MAPPO has the lowest proxy value (converging to 0.0026 with A*), while MADDPG shows the largest reduction from its initial values. The convergence stability plots (d) reveal that QMIX exhibits the highest training volatility (standard deviation up to 28), whereas MAPPO maintains the smoothest learning trajectory. The collision-penalty proxy combines collision types and is not a direct collision frequency.

## 6.4 Hypothesis Validation and Analysis

### 6.4.1 H1: Collision-Penalty Proxy Reduction

H1: A* guidance is associated with a lower collision-penalty proxy across different algorithms. Table 6.4 reports the corresponding proxy values for all three algorithms.

Table 6.4: H1 Validation: Collision-Penalty Proxy Analysis

| Algorithm | Base Collision Proxy | A* Collision Proxy | Change | Conclusion |
|---|---|---|---|---|
| MAPPO | 0.0032 | 0.0026 | -18.8% (better) | Supported |
| MADDPG | 0.0056 | 0.0049 | -12.5%(better) | Supported |
| QMIX | 0.0060 | 0.0047 | -21.7%(better) | Supported |
| **Average** | 0.00493 | 0.00407 | **-17.6% (better)** | **Strongly Supported** |

**Result: Strongly Supported.**

A* integration is associated with lower collision-penalty proxy values for all three algorithms (12.5%–21.7%) in these runs. This pattern is consistent with the interpretation that intermediate navigation targets may assist collision avoidance, but the single-seed evidence does not establish a general or causal effect. The proxy combines agent–obstacle and agent–agent collision penalties rather than separating them.

### 6.4.2 H2: Continuous vs Discrete Action Space

H2: Continuous-action algorithms benefit more from A* guidance than discrete-action algorithms. Table 6.5 compares the observed changes by action-space type.

Table 6.5: H2 Validation: Action Space Impact

| Algorithm | Action Type | Reward Change | Evaluation-Reward Change | Collision-Proxy Change | Conclusion |
|---|---|---|---|---|---|
| MADDPG | Continuous | +1.0% (better) | +6.6%(better) | -12.5%(better) | Supported |
| MAPPO | Continuous | -8.8%(worse) | -1.6% (worse) | -18.8% (better) | Mixed |
| QMIX | Discrete | -0.2%(worse) | -0.8%(worse) | -21.7%(better) | Not Supported |

**Result: Partially Supported.**

MADDPG shows the largest positive descriptive difference under A* guidance, possibly because experience replay allows path-guided trajectories to be reused during training. In contrast, QMIX shows a smaller descriptive difference despite being an off-policy method. These mechanisms were not separately tested, so the contrast should be treated as a possible explanation rather than a confirmed cause.

### 6.4.3 H3: On-Policy vs. Off-Policy Learning

H3: Off-policy algorithms benefit more from A* due to experience replay. Table 6.6 compares the observed outcomes for the on-policy and off-policy algorithms.

Table 6.6: H3 Validation: Learning Paradigm Comparison

| Algorithm | Type | Final-Window Evaluation Reward with A* | Evaluation-Reward Change | Conclusion |
|---|---|---|---|---|
| MADDPG | Off-Policy | 80.68 | +6.6% (better) | Strongly Supported |
| QMIX | Off-Policy | 68.19 | -0.8% (worse) | Not Supported |
| MAPPO | On-Policy | 77.87 | -1.6%(worse) | Not Supported |

**Result: Partially Supported.**

MADDPG shows a positive descriptive difference with A* guidance, whereas QMIX does not show the same pattern. This contrast indicates that off-policy learning alone is not sufficient to explain the observed results. Experience replay, continuous actions, and deterministic policies are possible explanations, but their individual effects were not tested.

### 6.4.4 Algorithm-Specific Analysis

**MADDPG Analysis:** MADDPG shows the largest positive descriptive difference in this comparison. Experience replay, deterministic policy execution, and the centralized critic are plausible explanations for this pattern, but these mechanisms were not measured or ablated separately.

**MAPPO Analysis:** The lower average episode reward ($-8.8\%$) observed alongside a lower collision-penalty proxy may be related to batch-size sensitivity in on-policy

learning. This mechanism was not tested separately and should be treated as a possible explanation.

**QMIX Analysis:** QMIX has a lower collision-penalty proxy but little change in the other reported metrics. The discrete action space is a possible explanation for this pattern, but its effect was not tested separately.

## 6.5 Discussion

### 6.5.1 Comparison with Lopez-Yepez et al.

Our findings extend the original study in several ways. Table 6.7 summarizes the differences in algorithms, framework, focus, and findings.

Table 6.7: Comparison with Original Study

| **Aspect** | **Lopez-Yepez et al.** | **My Study** |
|---|---|---|
| Algorithm | PPO/TD3 | MAPPO/MADDPG/QMIX |
| Framework | Custom | VMAS + BenchMARL |
| Focus | Single algorithm | Algorithm comparison |
| Key Finding | A* improves efficiency | Algorithm-dependent effects |

Lopez-Yepez et al. reported improvements in data efficiency and binary success rates. The evaluation-reward values in this study show only partial descriptive alignment with that conclusion, but the two quantities are not directly comparable. The observed evaluation-reward differences depend on the algorithm and parameter configuration.

In this experiment, MADDPG shows the strongest positive descriptive pattern with A* integration. In contrast, MAPPO has a lower average episode reward despite a lower collision-penalty proxy, and QMIX shows little overall difference. These results are partially consistent with the original study but do not confirm a general performance improvement.

### 6.5.2 Computational Overhead

Consistent with the observations of Lopez-Yepez et al., that "the computation of the A* heuristic can take a considerable amount of time during training," this experiment observed training-time overhead ranging from 32% to 97%. The algorithm-specific values are reported in Table 6.8.

Table 6.8: Training Time Overhead

| Algorithm | Base Time | A* Time | Overhead |
|---|---|---|---|
| MAPPO | 1h 12m | 2h 22m | +97% |
| MADDPG | 2h 10m | 3h 8m | +45% |
| QMIX | 2h 6m | 2h 47m | +33% |

## 6.6 Summary

Table 6.9 summarizes the outcomes of the three hypotheses evaluated in this chapter.

Table 6.9: Hypothesis Validation Summary

| Hypothesis | Core Statement | Result |
|---|---|---|
| H1 | Consistent collision reduction across algorithms | **Strongly Supported** |
| H2 | Continuous actions enable better A* utilization | Partially Supported |
| H3 | Off-policy learning amplifies A* benefits | Partially Supported |

This experiment extends the findings of Lopez-Yepez et al. to multiple MARL algorithms, revealing that:

1. The A*-guided runs have a 17.6% lower collision-penalty proxy on average across the three algorithms. This proxy does not directly measure collision frequency.

2. The results suggest possible MAPPO sensitivity to batch size, but the effect of batch size was not isolated from the other experimental differences.

3. MADDPG shows the strongest positive descriptive pattern in these runs. Its off-policy learning, continuous actions, and deterministic policy are plausible explanations rather than separately tested causes.

4. The discrepancy between Table 6.3 and Figure 6.1 suggests that averaged evaluation metrics alone may not fully capture the temporal effects of A* guidance. In particular, performance improvements for MAPPO and QMIX with A* become evident only in later training stages, highlighting the importance of analyzing learning dynamics in addition to final aggregated results.

These findings address the algorithm-generalization question and inform the design of the next experiment, which investigates whether Transformer architectures can better leverage A* guidance across different algorithms.

# Chapter 7

# Evaluation of Task-Adaptive Waypoint-Aware Transformer Policies

## 7.1 Introduction

This experiment compares the Fully Connected MLP Policy Baseline, the Standard Transformer, and the Task-Adaptive Waypoint-Aware Transformer (TA-WAT) configuration introduced in this thesis. The comparison evaluates the complete waypoint-input condition rather than a fundamentally different Transformer architecture. Under an experimental setup consistent with Chapter 6, we examine their observed navigation performance in obstacle-dense environments.

In addition, A*-generated waypoints are introduced as structured observation inputs. Both Transformer variants use the same input dimension and contain a five-slot waypoint field. For the Standard Transformer this field is zero-filled; for TA-WAT it contains the current A*-derived values $(\Delta x, \Delta y, d, d_x, d_y)$. All remaining Transformer architecture and training settings are shared. The comparison therefore evaluates the complete waypoint-input condition rather than any individual shared Transformer component.

## 7.2 Experimental Design

### 7.2.1 Research Hypotheses

- H1: The Standard Transformer outperforms the Fully Connected MLP Policy Baseline under A*-guided reinforcement learning.
- H2: Populating the shared five-slot waypoint field with A*-derived values, rather

than zeros, may produce descriptive differences in performance and training stability.

- H3: TA-WAT is expected to show descriptive performance differences relative to the Standard Transformer because its waypoint field contains A*-derived values rather than zeros. The comparison does not isolate the contribution of shared Transformer components, and any observed advantage may be metric-dependent and statistically uncertain.

These hypotheses examine whether the waypoint-input condition is associated with measurable descriptive differences when the remaining Transformer configuration is held constant.

### 7.2.2 Experimental Setup

This experiment largely follows the same training hyperparameters as those used in the previous chapters, ensuring consistency across experimental settings. However, as reported in Chapter 4, applying identical configurations to the Standard Transformer resulted in unstable training behavior. To address this issue, minor adjustments to efficiency-related parameters, such as batch size and optimization settings, were introduced to improve training stability.

Table 6.1 summarizes the general environment configuration and shared training parameters used in Chapter 7. The overall setup remains consistent with earlier experiments, including world size, sensor resolution, and evaluation protocol, allowing for fair comparison across architectures.

Table 7.1: Experiment Transformer Configuration Summary

| **Architecture Parameters** | | | |
|---|---|---|---|
| **Parameter** | **Value** | **Parameter** | **Value** |
| Actor $d_{model}$ | 256 | Critic $d_{model}$ | 128 |
| Actor Layers | 3 | Critic Layers | 2 |
| Actor Heads | 8 | Critic Heads | 4 |
| Actor $d_{ff}$ | 1024 | Critic $d_{ff}$ | 512 |
| Actor Dropout | 0.2 | Critic Dropout | 0.25 |
| **Optimization Features** | | | |
| Relative Positional Encoding | Yes | Local Attention | Yes |
| Gated FFN (Actor) | Yes | Local Attention Window | 3.0 units |
| Gated FFN (Critic) | No | Waypoint Field | Zero / A*-derived |
| **Training Hyperparameters** | | | |
| Batch Size | 30,000 frames | Learning Rate | $3 \times 10^{-5}$ |
| Minibatch Size | 4,096 | Weight Decay | $5 \times 10^{-5}$ |
| Minibatch Iterations | 12 | Warmup Steps | 8,000 |
| Clip Epsilon | 0.15 | Entropy Coefficient | 0.005 |
| GAE Lambda | 0.95 | Critic Coefficient | 0.5 |

Table 7.1 provides a detailed overview of the Transformer-based policy configurations, including architectural parameters, optimization features, and training hyperparameters. These settings are designed to balance model expressiveness and training stability.

Finally, Table 7.2 highlights the only intentional difference between the Standard Transformer and TA-WAT. Both configurations share the same input dimension, network structure, and training parameters. The Standard Transformer receives zeros in the five-slot waypoint field, whereas TA-WAT receives A*-derived values. Observed differences are therefore associated with the complete waypoint-input condition; the comparison does not isolate the effects of individual shared components.

Table 7.2: Differences Between the Standard Transformer and TA-WAT

| **Feature** | **Standard Transformer** | **TA-WAT** |
|---|---|---|
| Waypoint Field Values | Zero-filled | A*-derived |
| Input Dimension | Identical | Identical |
| Core Transformer Architecture | Identical | Identical |
| Training Parameters | Identical | Identical |

It is important to clarify that waypoint information serves **two distinct roles** during the training of TA-WAT. First, the shared five-slot field is populated with relative position ($\Delta x$, $\Delta y$), distance to the next waypoint, and unit direction vector ($d_x$, $d_y$); the corresponding field in the Standard Transformer contains zeros. Second, waypoint information is used for potential-based reward shaping in both variants. Thus, the controlled difference is the availability of non-zero waypoint observations, not the reward-shaping mechanism or an individual Transformer component.

A*-based reward shaping is removed during evaluation, but A*-derived waypoint observation features remain available to the policy. This means the evaluated policy still *perceives* waypoint guidance but is no longer *rewarded* for following it, allowing us to assess whether the policy has internalized the planning prior through its observation-level exposure rather than merely reacting to shaping rewards.

Table 7.3 summarizes which A*-derived signals are available to each configuration during training and evaluation.

### 7.2.3 Algorithm Selection

The same set of reinforcement learning algorithms as in the previous experiment is adopted in this study to ensure consistency and comparability of results. Specifically, MAPPO, MADDPG, and QMIX are selected to represent on-policy, off-policy actor-critic, and value-based learning paradigms, respectively. Their key characteristics are summarized in Table 6.2.

Table 7.3: Availability of A*-Derived Information During Training and Evaluation

| A*-Derived Component | MLP | Standard Transformer | TA-WAT |
|---|---|---|---|
| *Training Phase* | | | |
| A*-Based Reward Shaping | ✓ | ✓ | ✓ |
| Waypoint Observation Features | × | × | ✓ |
| *Evaluation Phase* | | | |
| A*-Based Reward Shaping | × | × | × |
| Waypoint Observation Features | × | × | ✓ |

The selection of algorithms follows exactly the same configuration as in the previous experiment, and no algorithm-specific modifications are introduced in this chapter. This design ensures that any observed performance differences can be attributed to architectural changes rather than differences in learning paradigms.

### 7.2.4 Baselines for Comparison

This study has two experimental groups. The first group centers on the MAPPO algorithm and examines how different model configurations affect navigation under A* guidance. We consider three configurations: MAPPO with the MLP baseline and A* guidance, MAPPO with the Standard Transformer and A* guidance, and MAPPO with TA-WAT and A* guidance. This chapter first analyzes single-agent navigation. The subsequent chapters investigate whether the observed benefits of A*-guided learning and TA-WAT can be maintained in cooperative dual-agent navigation tasks.

**Group 1: On-Policy MAPPO Architecture Comparison**

1. MAPPO-MLP-A*
2. MAPPO-Standard Transformer-A*
3. MAPPO-TA-WAT-A*

The second experimental group tests Transformer configurations across two additional MARL algorithms. Each algorithm uses either the Standard Transformer or TA-WAT. All configurations use A*-based guidance during training. During evaluation, A*-based reward shaping is disabled for all configurations. However, only the MLP and Standard Transformer configurations remove A*-derived information entirely; TA-WAT retains the 5-dimensional waypoint observation features in the policy input.

**Group 2: Off-Policy Transformer-Based Architecture Comparison**

1. MADDPG-Standard Transformer-A*
2. MADDPG-TA-WAT-A*

3. QMIX-Standard Transformer-A*

4. QMIX-TA-WAT-A*

## 7.3 Results

### 7.3.1 Group 1: Overall Performance Comparison

Table 7.4 presents the comprehensive performance metrics for the three MAPPO-based configurations. The results reveal a nuanced trade-off between convergence speed and final performance.

Table 7.4: Group 1: MAPPO Architecture Comparison Results

| Configuration | Avg Reward | Final Reward | Final-Window Evaluation Reward | Collision-Penalty Proxy | Convergence Iteration | CV |
|---|---|---|---|---|---|---|
| MAPPO-MLP-A* | 97.45 | 98.45 | 77.87 | **0.0026** | **18** | **0.078** |
| MAPPO-Transformer-A* | 78.69 | 87.15 | 62.06 | 0.0095 | 75 | 0.211 |
| MAPPO-Waypoint-Aware Transformer-A* | 87.27 | **104.17** | **87.07** | 0.0126 | 94 | 0.309 |

The MLP baseline policy achieves the earliest convergence (18 training iterations to reach 90% of final performance) and maintains the lowest relative reward variability (CV = 0.078, where CV denotes the coefficient of variation). However, in this descriptive comparison, TA-WAT produced the highest observed final reward (104.17) and final-window evaluation reward (87.07 reward units). These values represent results from the reported configuration and should not be interpreted as a statistically confirmed performance advantage.

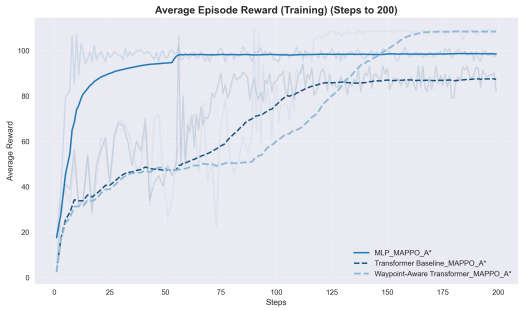
The Standard Transformer has lower observed values than both the MLP baseline and TA-WAT, including a final-window evaluation reward of 62.06 reward units and a higher CV of 0.211. This comparison establishes a descriptive difference under the tested configuration; it does not identify a causal effect of any shared Transformer component.

TA-WAT showed a 40.3% higher descriptive final-window evaluation-reward value than the Standard Transformer in this comparison (87.07 versus 62.06 reward units). Because relative positional encoding, gated feed-forward networks, and local attention were shared by both variants and were not independently ablated, the observed difference should be associated with the complete waypoint-input condition rather than attributed to any of these components. The difference was not statistically confirmed in the five-seed analysis.

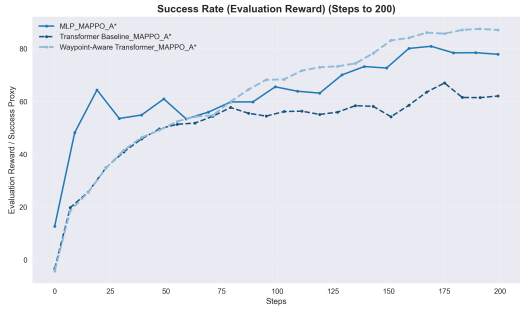
### 7.3.2 Group 1: Learning Dynamics Analysis

Figure 7.1 illustrates the training progression over 200 training iterations.

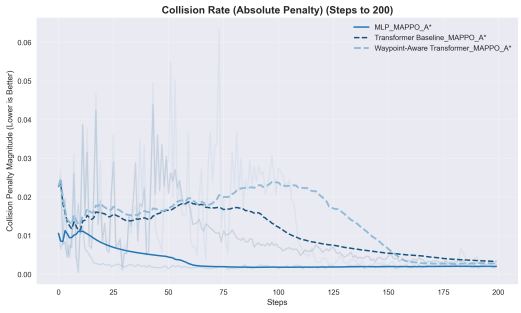
The average reward curves show clear differences between configurations. The MLP Policy Baseline learns quickly at the early stage of training. It reaches near-optimal



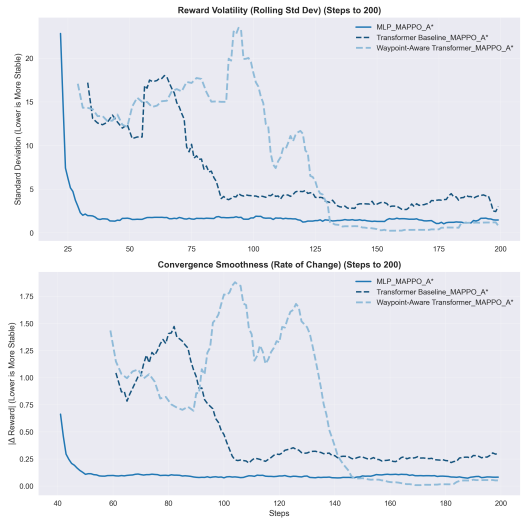

(a) Average Episode Reward




(b) Evaluation Reward (legacy embedded plot label: "Success Rate")




(c) Collision-Penalty Proxy




(d) Convergence Stability

Figure 7.1: Group 1: Training dynamics comparison of MAPPO with different architectures. Evaluation disables A*-based reward shaping; waypoint observations remain accessible only for the Task-Adaptive Waypoint-Aware Transformer variant.

performance within about 25 training iterations. In contrast, the Transformer-based models improve more slowly over time. Their performance increases gradually throughout training. TA-WAT continues to improve in the later stage and surpasses the MLP baseline in final reward after around 125 training iterations. This result indicates slower initial learning but a higher observed final reward in this run.

The collision-penalty proxy shows that the MLP Policy Baseline maintains consistently low values throughout training (converging to 0.0026), while the Transformer models have higher proxy values during exploration before gradually decreasing. The convergence stability plots show the lowest observed variability for the MLP Policy Baseline, while TA-WAT shows higher variance during the middle training phase (steps 75–125) before stabilizing. The collision proxy is not a direct collision rate.

### 7.3.3 Group 2: Overall Performance Comparison

Table 7.5 presents the cross-algorithm comparison of the Transformer configurations. TA-WAT shows mixed but generally positive descriptive results relative to the Standard Transformer. For MADDPG, TA-WAT has higher average reward, final reward, and final-window evaluation-reward values, together with lower collision-penalty proxy and variability values and earlier observed convergence. For QMIX, it has slightly higher average and final rewards and earlier observed convergence, but the final-window evaluation reward is lower and the collision-penalty proxy is marginally higher. Therefore, the evidence suggests algorithm-dependent descriptive differences rather than consistent outperformance across all metrics.

Table 7.5: Group 2: Waypoint-Aware Transformer Across Algorithms

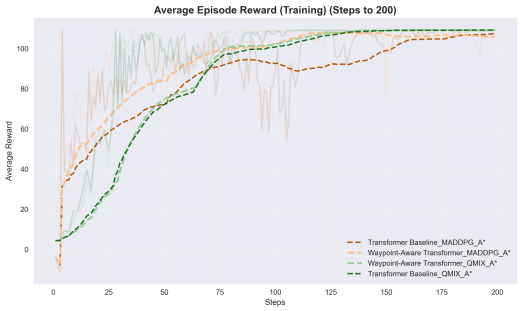
| Configuration | Avg Reward | Final Reward | Final-Window Evaluation Reward | Collision-Penalty Proxy | Convergence Iteration | CV |
|---|---|---|---|---|---|---|
| MADDPG-Transformer-A* | 93.06 | 103.25 | 59.70 | 0.0181 | 78 | 0.210 |
| MADDPG-Waypoint-Aware Transformer-A* | **100.02** | **106.75** | 64.27 | 0.0121 | 65 | **0.171** |
| QMIX-Transformer-A* | 99.55 | 109.11 | **77.11** | **0.0043** | 69 | 0.215 |
| QMIX-Waypoint-Aware Transformer-A* | 100.61 | 109.19 | 73.21 | 0.0045 | **61** | 0.214 |

For MADDPG, TA-WAT has higher descriptive values across several metrics. The average reward is 7.5% higher (93.06 $\rightarrow$ 100.02), while the collision-penalty proxy is 33.1% lower (0.0181 $\rightarrow$ 0.0121). Training stability is also improved, with the coefficient of variation (CV) reduced from 0.210 to 0.171. In addition, convergence is achieved more rapidly, with the Task-Adaptive Waypoint-Aware model reaching 90% performance 13 training iterations earlier (78 $\rightarrow$ 65).

For QMIX, the results are mixed. The average reward increases slightly from 99.55 to 100.61, and convergence is reached earlier, from training iteration 69 to 61. However, the final-window evaluation reward decreases from 77.11 to 73.21 reward units, and the collision-penalty proxy increases slightly from 0.0043 to 0.0045. Thus, TA-WAT shows only limited and metric-dependent descriptive differences for QMIX.

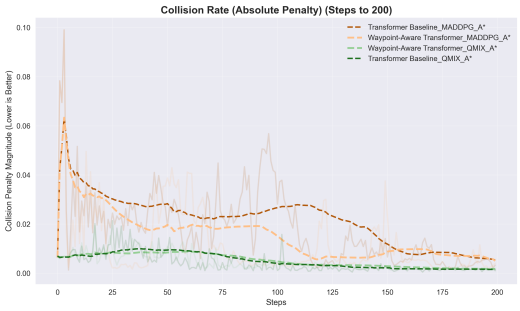
### 7.3.4 Group 2: Learning Dynamics Analysis

Figure 7.2 illustrates the training progression for Group 2 experiments.

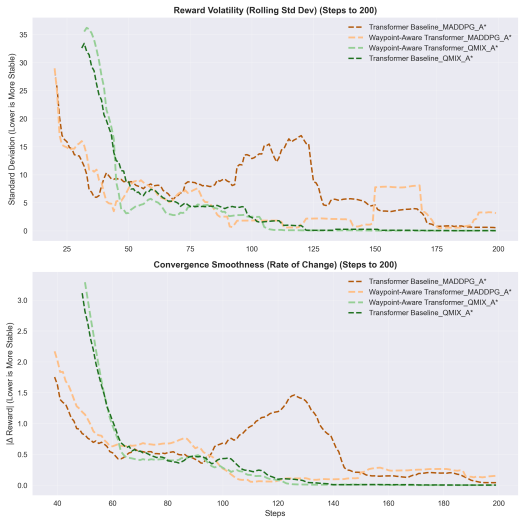



(a) Average Episode Reward


(b) Evaluation Reward (legacy embedded plot label: "Success Rate")




(c) Collision-Penalty Proxy




(d) Convergence Stability

Figure 7.2: Group 2: Training dynamics comparison of MADDPG and QMIX with Transformer-based architectures. Evaluation disables A*-based reward shaping; waypoint observations remain accessible only for the Task-Adaptive Waypoint-Aware Transformer variants.

The reward curves indicate that all four configurations achieve comparable final performance, with average rewards in the range of 100–110. QMIX-based approaches exhibit slightly faster initial learning during the early training phase.

The collision-penalty proxy trends show differences between algorithm types. QMIX maintains consistently low proxy values from early training stages, whereas the MADDPG configurations have higher values that decrease gradually over time. Because the proxy combines collision penalties, it is not a direct measure of collision frequency or safety.

The convergence stability analysis shows that the Waypoint-Aware Transformer

improves training smoothness for MADDPG, with the coefficient of variation (CV) reduced from 0.210 to 0.171, while having minimal impact on QMIX stability (CV: $0.215 \rightarrow 0.214$). In the reported runs, the waypoint-aware configuration showed a larger descriptive stability difference for MADDPG than for QMIX. However, the experiment does not establish that this difference was caused by the continuous action space or by any individual architectural component.

## 7.4 Hypothesis Validation and Analysis

### 7.4.1 H1: Transformer vs. MLP Baseline Performance

H1: Transformer-based policy networks outperform MLP-based networks under A*-guided reinforcement learning.

Table 7.6 reports the MAPPO architecture comparison used to evaluate H1.

Table 7.6: H1 Validation: Architecture Comparison (MAPPO)

| Architecture | Avg Reward | Final Reward | Final-Window Evaluation Reward | Convergence Iteration | Conclusion |
|---|---|---|---|---|---|
| MLP(Baseline) | 97.45 | 98.45 | 77.87 | 18 | Baseline |
| Standard Transformer | 78.69 (-19.3%) | 87.15 (-11.5%) | 62.06 (-20.3%) | 75 | Not Supported |
| TA-WAT | 87.27 (-10.4%) | **104.17 (+5.8%)** | **87.07 (+11.8%)** | 94 | Partially Supported |

**Result: Partially supported descriptively; not statistically confirmed.**

The experimental results present a nuanced picture. The Standard Transformer has a 19.3% lower average-reward value and a 20.3% lower final-window evaluation-reward value than the MLP baseline. TA-WAT has the highest observed final reward (+5.8%) and final-window evaluation-reward value (+11.8%) in this descriptive comparison. These differences are associated with the complete waypoint-input condition, although the evidence should be interpreted cautiously because the statistical analysis does not establish significance under the current five-seed evaluation.

### 7.4.2 H2: Waypoint-Input Condition Effects

H2: Populating the shared five-slot waypoint field with A*-derived values rather than zeros may change descriptive performance and stability.

Table 7.7 reports the descriptive differences between the Standard Transformer and TA-WAT used to evaluate H2.

Table 7.7: H2 Evaluation: Descriptive Differences Between Standard and Waypoint-Aware Transformer Configurations

| Algorithm | Reward Change | Evaluation-Reward Change | Collision-Proxy Change | Conv. Change | CV Change | Conclusion |
|---|---|---|---|---|---|---|
| MAPPO | +10.9% | +40.3% | +32.6% ↓ | +25.3% ↓ | +46.4% ↓ | Mixed descriptive difference; not statistically confirmed |
| MADDPG | +7.5% | +7.7% | -33.1% ↑ | -16.7% ↑ | -18.6% ↑ | Positive descriptive difference; not statistically confirmed |
| QMIX | +1.1% | -5.1% | +4.7% ↓ | -11.6% ↑ | -0.5% ↑ | Mixed descriptive difference; not statistically confirmed |

For Table 7.7, positive values consistently indicate a favourable performance change

and negative values indicate an unfavourable change. Reward and evaluation-reward changes use (TA-WAT − Standard)/Standard; collision-proxy, convergence-iteration, and CV changes use (Standard − TA-WAT)/Standard because lower values are preferable for those metrics. All entries are descriptive and are not statistically confirmed by the five-seed tests.

**Result: Partially supported based on descriptive trends; not statistically confirmed.**

The waypoint-aware configurations show positive descriptive differences in selected metrics, but the patterns are not uniform across algorithms or metrics. Because all Holm-adjusted tests are non-significant, the results do not establish a reliable performance advantage. They also do not identify effects attributable to individual Transformer components.

- **MAPPO:** average reward increases by 10.9%, and the final-window evaluation reward increases by 40.3%.
- **MADDPG:** average reward increases by 7.5%, the collision-penalty proxy decreases by 33.1%, and training stability improves by 18.6%.
- **QMIX:** average reward increases by 1.1%, and convergence is achieved 11.6% faster.

### 7.4.3 H3: Cross-Algorithm Generalization

H3: The effectiveness of Transformer-based architectures is consistent across different reinforcement learning algorithms.

Table 7.8 compares the observed architecture differences across the three algorithms.

Table 7.8: H3 Validation: Cross-Algorithm Comparison

| Algorithm | Learning Type | Reward Improvement | Convergence Speedup | Conclusion |
|---|---|---|---|---|
| MAPPO | On-Policy | +10.9% | -25.3% (slower) | Positive descriptive trend |
| MADDPG | Off-Policy | +7.5% | +16.7% | Positive descriptive trend |
| QMIX | Off-Policy | +1.1% | +11.6% | Limited descriptive trend |

*Note: Cross-algorithm generalization was not statistically confirmed by the five-seed analysis.*

**Result: Partially supported based on descriptive trends; not statistically confirmed.**

The Task-Adaptive Waypoint-Aware Transformer shows positive descriptive reward changes for MAPPO, MADDPG, and QMIX, but the magnitude varies substantially and the benefits are not uniform across all metrics. Positive descriptive reward differences were observed for the waypoint-aware configuration in MAPPO, MADDPG, and QMIX, but their magnitude varied and other metrics showed mixed results. These observations do not establish statistically significant cross-algorithm generalization.

### 7.4.4 Algorithm-Specific Analysis

**MAPPO Analysis:** The reported runs show the most pronounced trade-off between configurations. The MLP baseline reaches its convergence criterion 5.2× earlier and has lower variability. TA-WAT has an 11.8% higher observed final-window evaluation-reward value. Because this difference was not statistically confirmed, application-level recommendations remain tentative.

**MADDPG Analysis:** MADDPG shows the largest observed descriptive stability improvement among the tested algorithms. The coefficient of variation decreases from 0.210 to 0.171, suggesting improved training smoothness in this configuration. In the reported MADDPG configuration, the coefficient of variation decreased from 0.210 to 0.171. This is a descriptive observation and does not establish that the change was caused by attention-enhanced representations, continuous control, or any individual Transformer component.

**QMIX Analysis:** QMIX has consistently low collision-penalty proxy values across the reported configurations (approximately 0.004). Because this proxy combines agent–obstacle and agent–agent collision penalties and is not a direct collision rate, these values do not establish collision frequency or a causal safety effect of value decomposition.

## 7.5 Statistical Significance Analysis

To further strengthen the validity of the conclusions regarding the effectiveness of the Waypoint-Aware Transformer, paired $t$-tests with Holm-Bonferroni correction are conducted across multiple random seeds. This analysis accounts for the stochastic nature of reinforcement learning by controlling for inter-seed variability and the family-wise error rate.

### 7.5.1 Methodology

Each algorithm–architecture combination is evaluated across 5 random seeds (26, 38, 48, 66, 96). For each algorithm, a two-tailed paired $t$-test is performed comparing the evaluation rewards of the Standard Transformer and TA-WAT. Since three simultaneous comparisons are conducted (one per algorithm), Holm-Bonferroni correction is applied to control the family-wise error rate at $\alpha = 0.05$. Cohen's $d$ is reported as a standardized effect size to quantify the practical significance of observed differences.

### 7.5.2 Results

Table 7.9 presents the paired comparison results for evaluation reward.

Table 7.10 provides the full comparison across all metrics.

Table 7.9: Paired $t$-test results for evaluation reward across 5 random seeds.

| Algorithm | Standard Transformer | Waypoint-Aware Transformer | $\Delta$ | 95% CI | Raw $p$ | Holm-adjusted $p$ | Cohen's $d$ |
|---|---|---|---|---|---|---|---|
| MAPPO | 55.31 | 61.65 | +6.33 | [−8.50, 21.16] | 0.300 | 0.465 | 0.53 |
| QMIX | 26.67 | 36.44 | +9.78 | [−5.80, 25.36] | 0.156 | 0.465 | 0.78 |
| MADDPG | 9.28 | 28.05 | +18.77 | [−16.54, 54.08] | 0.212 | 0.465 | 0.66 |

Table 7.10: Full comparison across training reward and the collision-penalty proxy.

| Algorithm | Architecture | Training Reward | Collision-Penalty Proxy |
|---|---|---|---|
| MAPPO | Standard Transformer | $104.92 \pm 8.74$ | $0.0020 \pm 0.0011$ |
| MAPPO | TA-WAT | $108.87 \pm 0.42$ | $0.0028 \pm 0.0013$ |
| QMIX | Standard Transformer | $109.17 \pm 0.22$ | $0.0018 \pm 0.0009$ |
| QMIX | TA-WAT | $109.31 \pm 0.18$ | $0.0011 \pm 0.0003$ |
| MADDPG | Standard Transformer | $103.82 \pm 9.17$ | $0.0156 \pm 0.0137$ |
| MADDPG | TA-WAT | $95.77 \pm 23.18$ | $0.0243 \pm 0.0313$ |

Table 7.11 shows per-seed evaluation rewards.

Table 7.11: Per-seed evaluation rewards across 5 random seeds.

| Configuration | 38 | 26 | 48 | 66 | 96 |
|---|---|---|---|---|---|
| MAPPO Standard Transformer | 64.9 | 65.4 | 15.5 | 23.0 | 107.7 |
| MAPPO TA-WAT | 92.5 | 65.8 | 17.7 | 24.3 | 107.8 |
| QMIX Standard Transformer | 75.0 | 17.9 | 15.0 | −1.8 | 27.2 |
| QMIX TA-WAT | 82.8 | 18.5 | 16.8 | 29.6 | 34.6 |
| MADDPG Standard Transformer | 46.7 | −23.5 | 16.0 | 7.3 | −0.0 |
| MADDPG TA-WAT | 46.5 | 22.7 | 60.3 | 30.6 | −19.9 |

### 7.5.3 Interpretation

Paired $t$-tests were conducted across matched random seeds ($n = 5$).

**Observed directional trend.** Across the five matched random seeds, the mean evaluation rewards are higher for the Waypoint-Aware Transformer than for the Transformer baseline in all three algorithms. However, the Holm-adjusted p-values are 0.465 and all 95% confidence intervals include zero, so the observed differences are not statistically significant.

**Effect size analysis.** The effect sizes suggest that the trend may be practically relevant, but the current sample size is too small to draw firm conclusions. Future work with more random seeds and broader evaluation is needed to determine whether these effects are reliable.

**Statistical power limitation.** With $n = 5$ evaluation runs, statistical power is limited. Each run uses the same held-out set of 20 Easy maps (maps 31–50), while training randomness differs across the matched seeds. Therefore, non-significant results should be interpreted with caution. This does not necessarily imply the absence of an

effect, but rather that there is insufficient evidence to establish statistical significance under the current experimental setup.

Overall, the results show positive descriptive mean differences for the Waypoint-Aware Transformer, but the five-seed analysis does not establish a statistically reliable advantage. Further validation with larger sample sizes is required.

### 7.5.4 Robustness Analysis

Performance improvements are most pronounced in unstable or failure-prone seeds. In several cases, the baseline collapses or produces degenerate policies, while TA-WAT maintains stable performance or recovers successfully. These runs show an exploratory robustness pattern in some seeds. However, robustness was not evaluated through a separate powered analysis, so the results do not establish robustness as the primary benefit of the waypoint-aware configuration.

### 7.5.5 Summary of Statistical Findings

Although statistical significance is not achieved, the observed mean differences and some robustness patterns suggest a possible practical benefit of the Waypoint-Aware Transformer. These findings require validation with larger sample sizes.

## 7.6 Discussion

### 7.6.1 Architecture-Guidance Interaction

In this study, the Standard Transformer and TA-WAT use the same input dimension and the same core Transformer architecture. The shared five-slot waypoint field is zero-filled for the Standard Transformer and contains A*-derived values for TA-WAT. In the reported comparison, the Standard Transformer produced a 19.3% lower average-reward value than the MLP baseline. Because the model receives only current-timestep observation tokens, this result concerns relational modeling within the current observation and should not be interpreted as evidence about temporal or sequential memory. This descriptive result may reflect unnecessary model complexity under the tested configuration, but the underlying mechanism was not separately tested.

The Task-Adaptive Waypoint-Aware Transformer shows positive descriptive differences in selected metrics under the complete waypoint-input condition. However, the statistical analysis does not establish a significant advantage. Relative positional encoding, gated FFN, and local attention were shared by both Transformer variants and were not independently evaluated through ablation. Therefore, the reported differences cannot be attributed separately to any of these components.

### 7.6.2 Convergence Iteration vs. Final Performance

The results highlight a trade-off in architecture selection, summarized in Table 7.12.

Table 7.12: Convergence Iteration vs. Final Performance Trade-off

| Architecture | Convergence Iteration | Final-Window Evaluation Reward | Recommendation |
|---|---|---|---|
| MLP baseline | 18 (earliest) | 77.87 | Sample-limited scenarios |
| Waypoint-Aware Transformer | 94 (5.2× later) | 87.07 (+11.8%) | Performance-critical applications |

The MLP baseline reaches 90% of its final performance within 18 training iterations. TA-WAT requires 94 training iterations to reach the same level. This corresponds to a 5.2× difference in convergence iterations. However, TA-WAT achieves a higher observed final-window evaluation reward: 87.07 reward units, compared with 77.87 for the MLP. This represents a descriptive relative difference of 11.8%. These results suggest a tentative selection criterion. The MLP appears more suitable for sample-limited scenarios because it reaches the convergence criterion much earlier in the observed experiments. TA-WAT may be preferable in settings that prioritize final evaluation reward, but this recommendation should be treated cautiously because the observed final-performance difference has not been statistically confirmed.

### 7.6.3 Computational Overhead

Training-time values for the three MAPPO configurations are reported in Table 7.13.

Table 7.13: Training Time Comparison (MAPPO)

| Architecture | Training Time | Overhead vs. MLP |
|---|---|---|
| MLP | 2h 23m | baseline |
| Standard Transformer | 2h 48m | +17.4% |
| Waypoint-Aware Transformer | 2h 13m | -6.9% |

In the reported MAPPO runs, TA-WAT required less wall-clock training time than both the MLP baseline and the Standard Transformer. This observation should not be interpreted as evidence that waypoint inputs reduce training cost, because training-time differences were not separately controlled or statistically tested.

## 7.7 Summary

Table 7.14 summarizes the outcomes of the hypotheses evaluated in this chapter.

This experiment investigated the impact of Transformer-based architectures on multi-agent navigation with A* waypoint guidance. Through systematic comparison of MLP baseline, standard Transformer, and Task-Adaptive Waypoint-Aware Transformer across three MARL algorithms, we established the following key findings:

Table 7.14: Hypothesis Validation Summary

| Hypothesis | Core Statement | Result |
|---|---|---|
| H1 | Transformer baseline outperforms MLP baseline under A* guidance | **Mixed descriptive evidence; not statistically confirmed** |
| H2 | Waypoint-input condition changes Transformer performance | **Positive but mixed descriptive trends; not statistically confirmed** |
| H3 | Benefits generalize across RL algorithms | **Descriptive trends only; cross-algorithm generalization not confirmed** |

1. The Standard Transformer has a 19.3% lower average-reward value and a 20.3% lower final-window evaluation-reward value than the MLP baseline under the tested configuration. This descriptive comparison does not establish the effect of model complexity or any individual Transformer component.

2. The waypoint-input condition shows positive descriptive differences in selected metrics, with average-reward differences ranging from 1.1% in QMIX to 10.9% in MAPPO and higher final-window evaluation-reward values in some configurations. However, the differences are not consistent across all metrics, and the paired t-tests with Holm-Bonferroni correction do not establish statistical significance. The results therefore do not establish a general performance advantage.

3. The reported runs show a descriptive trade-off between convergence iteration and final performance. The MLP reaches its convergence criterion 5.2× earlier but has an 11.8% lower final-window evaluation-reward value than TA-WAT. Because the final-performance difference was not statistically confirmed, this finding provides only tentative guidance for architecture selection.

4. Positive descriptive reward differences were observed for the waypoint-aware configuration across MAPPO, MADDPG, and QMIX, but improvements were not consistent across all metrics. The paired five-seed tests did not establish statistical significance. Therefore, cross-algorithm generalization is not confirmed and requires additional evaluation.

These findings provide actionable guidance for practitioners: when deploying attention-based architectures for navigation with planning guidance, the tested results suggest that explicit waypoint observations may be useful in some algorithm configurations, but their reliable benefit and the contributions of individual architectural components remain unconfirmed.

# Chapter 8

# Feasibility and Limits of A* Guidance in Dual-Agent Navigation

## 8.1 Introduction

Chapter 6 examined the generalization of A* integration across different MARL algorithms in single-agent navigation scenarios. The findings showed lower collision-penalty proxy values with A* guidance while other performance metrics exhibited algorithm-dependent differences. Lopez-Yepez et al. [4] specifically noted that "exploring the application of the proposed method in scenarios involving multiple autonomous agents" remains an open research direction.

This experiment directly addresses this gap by extending the investigation to genuine multi-agent scenarios with two agents operating in a shared environment. It is important to note that this chapter focuses exclusively on the pathfinding behavior of two agents and does not aim to model or analyze explicit cooperative or competitive strategies. The multi-agent setting is therefore used to examine navigation complexity arising from inter-agent interaction and collision avoidance, rather than coordinated task-level cooperation or adversarial behavior.

## 8.2 Experimental Design

### 8.2.1 Research Hypotheses

- H1: A* guidance maintains lower collision-penalty proxy values in multi-agent scenarios.

- H2: Training strategies developed for single-agent A*-guided reinforcement learning are expected to transfer only partially to dual-agent navigation, because independent A* guidance may introduce inter-agent path conflicts and alter the balance between learning performance, collision avoidance, and generalization.
- H3: Evaluation rewards improve more with A* guidance than training rewards in the reported runs.

### 8.2.2 Experimental Setup

The experimental configuration follows the settings established in previous chapters, with the critical modification of increasing the number of agents from one to two. Table 8.1 reports the complete configuration used in this experiment.

Table 8.1: Experiment Configuration Summary

| **Parameter** | **Value** | **Parameter** | **Value** |
|---|---|---|---|
| World Size | $10 \times 10$ units | Training Frames | 6,000,000 |
| Grid Resolution | $50 \times 50$ cells | Parallel Environments | 120 |
| **Number of Agents** | **2** | Batch Size | 30,000 frames |
| Agent Radius | 0.15 units | Learning Rate | $5 \times 10^{-4}$ |
| LiDAR Rays | 16 (360°) | Random Seed | 38 |
| Max Episode Steps | 500 | Evaluation Episodes | 100 |
| Map Difficulty | Easy (50 maps) | | |

### 8.2.3 Algorithm Selection

The same three algorithms from Chapter 6 are evaluated to enable direct comparison between single-agent and multi-agent scenarios. Their main characteristics are summarized in Table 8.2.

Table 8.2: Algorithm Characteristics

| **Property** | **MAPPO** | **MADDPG** | **QMIX** |
|---|---|---|---|
| Learning Type | On-Policy | Off-Policy | Off-Policy |
| Action Space | Continuous | Continuous | Discrete |
| Policy Type | Stochastic | Deterministic | Value-Based |

### 8.2.4 Baselines for Comparison

A total of six experiments are conducted, identical in structure to Chapter 6:

1. MAPPO-A*: MAPPO with A* waypoint guidance

2. MAPPO-Base: MAPPO without A* guidance
3. MADDPG-A*: MADDPG with A* waypoint guidance
4. MADDPG-Base: MADDPG without A* guidance
5. QMIX-A*: QMIX with A* waypoint guidance
6. QMIX-Base: QMIX without A* guidance

## 8.3 Results

### 8.3.1 Overall Performance Comparison

Table 8.3 reports the reward, collision, convergence, and training-time values for all six dual-agent configurations.

Table 8.3: Performance Comparison of Different Algorithms (2 Agents)

| Configuration | Avg Reward | Final-Window Evaluation Reward | Collision-Penalty Proxy | Convergence Iteration | Training Time |
|---|---|---|---|---|---|
| MADDPG-A* | 1596.06 | **1014.71** | 0.0180 | 131 | 2h 53m |
| MADDPG-Base | 1504.23 | 423.62 | **0.0155** | **85** | 2h 10m |
| MAPPO-A* | 1440.35 | 938.31 | 0.0215 | 38 | 3h 14m |
| MAPPO-Base | **1902.80** | 668.16 | 0.0209 | 61 | **2h 9m** |
| QMIX-A* | 1373.96 | 834.04 | 0.0176 | 126 | 3h 44m |
| QMIX-Base | 1583.14 | 323.27 | 0.0202 | 114 | 2h 15m |

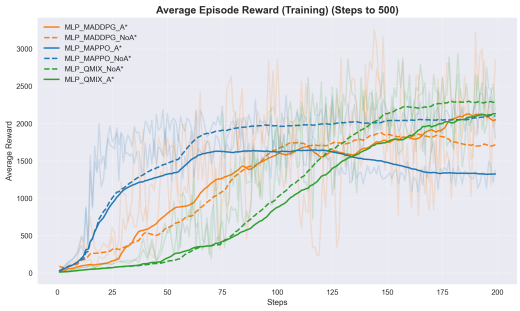
The results reveal a notable descriptive divergence between training reward and evaluation reward when comparing A* and baseline configurations. In the reported dual-agent runs, A*-guided configurations have higher evaluation-reward values but do not consistently have higher training-reward values. These differences were not subjected to a multi-seed significance test.

MAPPO-Base has the highest observed training reward (1902.80), compared with 1440.35 for its A* counterpart. MADDPG-A* has the highest observed evaluation reward (1014.71), compared with 423.62 for its baseline. These are descriptive differences and do not establish that A* guidance causes more conservative or more generalizable policies.

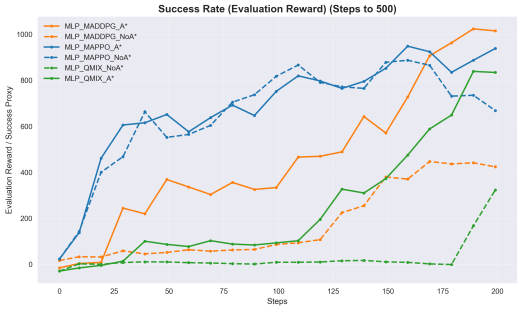
### 8.3.2 Learning Dynamics Analysis

Figure 8.1 shows training progress over 200 training iterations.

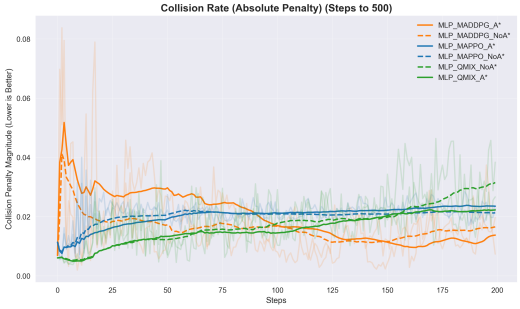
The average reward curves (a) demonstrate distinct learning patterns compared to single-agent scenarios. MAPPO-Base shows the fastest reward growth, achieving stable high performance around training iteration 75. In contrast, MAPPO-A* exhibits slower initial learning and plateaus at a lower reward level, suggesting that A* guidance may constrain exploration in multi-agent settings. QMIX configurations show



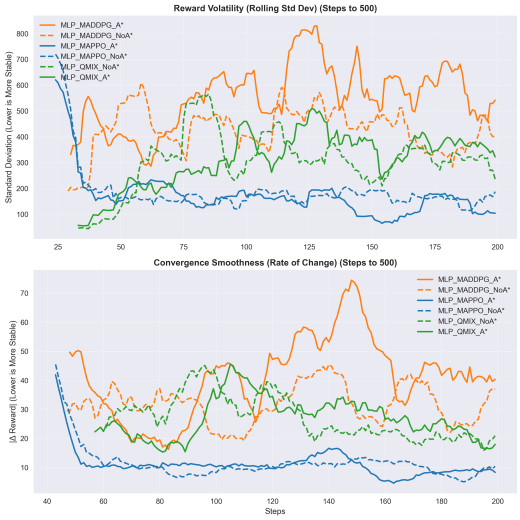

(a) Average Episode Reward




(b) Evaluation Reward (legacy embedded plot label: "Success Rate")




(c) Collision-Penalty Proxy




(d) Convergence Stability

Figure 8.1: Training dynamics comparison of different algorithms for two-agent navigation, evaluated without A* guidance.

gradual improvement throughout training, with the A* variant maintaining slightly lower rewards.

The evaluation reward curves (b) tell a different story. All A* variants have higher evaluation-reward values than their baselines. MADDPG-A* reaches the highest final evaluation reward (approximately 1000), while QMIX-A* shows the largest descriptive improvement ratio. This divergence between training and evaluation metrics is a key finding in the reported multi-agent runs.

The collision-penalty proxy (c) reveals a different pattern from the single-agent scenarios: A* guidance does not consistently reduce this proxy in multi-agent settings. MADDPG-A* and MAPPO-A* show higher values than their baselines, while only QMIX-A* shows a lower value. This proxy combines agent–obstacle and agent–agent collision penalties; it is not a direct collision rate and does not separate collision types. Inter-agent path conflict is therefore only a plausible explanation for the observed pattern.

The convergence stability plots (d) indicate that MAPPO configurations maintain the smoothest learning trajectories, consistent with findings from previous chapters. MADDPG shows higher volatility, particularly during middle training stages.

## 8.4 Hypothesis Validation and Analysis

### 8.4.1 H1: Collision-Penalty Proxy in Multi-Agent Scenarios

H1: A* guidance maintains lower collision-penalty proxy values in multi-agent scenarios. Table 8.4 reports the proxy values used to evaluate this hypothesis.

Table 8.4: H1 Validation: Collision-Penalty Proxy Analysis (2 Agents)

| Algorithm | Base Proxy | A* Proxy | Change | Conclusion |
|---|---|---|---|---|
| MAPPO | 0.0209 | 0.0215 | +2.9% (worse) | Not Supported |
| MADDPG | 0.0155 | 0.0180 | +16.1%(worse) | Not Supported |
| QMIX | 0.0202 | 0.0176 | -12.9% (better) | Supported |
| **Average** | 0.0189 | 0.0190 | **+0.9%** | **Not Supported** |

**Result: Not Supported.**

Contrary to the single-agent findings, A* guidance does not consistently reduce the collision-penalty proxy in two-agent scenarios. MAPPO and MADDPG show higher proxy values with A* (+2.9% and +16.1%, respectively), while QMIX shows a lower value (-12.9%).

One plausible explanation for this finding is the independent nature of A* path computation. Each agent receives an individually planned path that does not explicitly account for the other agent's trajectory. The paths may therefore overlap at intersections

or narrow passages, potentially increasing the risk of inter-agent conflict. However, collision types and explicit conflicts between planned paths were not measured separately. Therefore, inter-agent path conflict should be interpreted as a plausible explanation rather than a directly established cause of the observed collision pattern.

### 8.4.2 H2: Diminished A* Effectiveness in Multi-Agent Settings

H2: Multi-agent coordination challenges diminish the effectiveness of A* guidance. Table 8.5 compares the training-reward values used to evaluate this hypothesis.

Table 8.5: H2 Validation: Training Reward Comparison

| Algorithm | Base Reward | A* Reward | Change | Conclusion |
|---|---|---|---|---|
| MAPPO | 1902.80 | 1440.35 | -24.3% | Strongly Supported |
| MADDPG | 1504.23 | 1596.06 | +6.1% | Not Supported |
| QMIX | 1583.14 | 1373.96 | -13.2% | Supported |

**Result: Partially Supported.**

The training-reward analysis shows mixed descriptive results. MAPPO has the largest observed reduction (-24.3%) with A* guidance, while QMIX also has a lower value (-13.2%). MADDPG is the only tested algorithm with a higher training-reward value under A* guidance (+6.1%).

The coordination-overhead hypothesis is descriptively supported for MAPPO and QMIX but not for MADDPG. Algorithm-specific update mechanisms are possible explanations for this difference, but they were not tested separately.

### 8.4.3 H3: Evaluation-Reward Differences

H3: A* guidance improves evaluation rewards more than training rewards. Table 8.6 compares the observed training- and evaluation-reward changes.

Table 8.6: H3 Evaluation: Evaluation-Reward vs. Training-Reward Differences

| Algorithm | Train Change | Eval Change | Eval > Train | Conclusion |
|---|---|---|---|---|
| MAPPO | -24.3% | +40.4% | Yes | Strongly Supported |
| MADDPG | +6.1% | +139.5% | Yes | Strongly Supported |
| QMIX | -13.2% | +158.0% | Yes | Strongly Supported |

**Result: Strongly Supported.**

All three algorithms show larger descriptive relative changes in evaluation reward than in training reward. QMIX has a 13.2% lower training-reward value and a 158.0% higher evaluation-reward value. This pattern does not by itself confirm improved policy generalization, because no direct generalization measure or multi-seed significance test was conducted.

One possible explanation is that waypoint guidance provides a more structured evaluation trajectory, but this mechanism was not measured separately.

### 8.4.4 Algorithm-Specific Analysis

**MAPPO Analysis:** MAPPO exhibits the largest training–evaluation gap. The training-reward value is 24.3% lower and the evaluation-reward value is 40.4% higher under A* guidance. A possible explanation involves changes in exploration and sample diversity, but these mechanisms were not measured separately.

**MADDPG Analysis:** MADDPG shows positive descriptive differences in both training (+6.1%) and evaluation (+139.5%) reward. Experience replay, deterministic policies, and the centralized critic are plausible explanations, but their contributions were not tested separately.

**QMIX Analysis:** QMIX shows the largest descriptive evaluation-reward increase (+158.0%) despite a lower training reward (-13.2%). QMIX is also the only algorithm with a lower collision-penalty proxy under A* (-12.9%). Because collision types and path conflicts were not measured separately, these observations do not establish a coordination mechanism or an effect of the action-space design.

## 8.5 Discussion

### 8.5.1 Single-Agent vs Multi-Agent Comparison

Table 8.7 summarizes the descriptive differences between the single-agent and dual-agent experiments.

Table 8.7: Comparison: Single-Agent vs Multi-Agent A* Effects

| Metric | Single-Agent | Multi-Agent (2) | Trend |
|---|---|---|---|
| Collision-Penalty Proxy Change | -17.6% (avg) | +0.5% (avg) | Reversed |
| Training Reward | Mixed effects | Generally decreased | Degraded |
| Evaluation Reward | Moderate improvement | Large improvement | Enhanced |
| MADDPG Compatibility | High | High | Consistent |

The transition from single-agent to multi-agent scenarios reveals fundamental changes in how A* guidance affects learning:

1. Collision-penalty proxy reversal: A* is associated with a 17.6% lower proxy in the single-agent scenarios but a slight increase (+0.5% on average) in the multi-agent scenarios. This reversal may plausibly be related to conflicts between independently computed A* routes; however, explicit path conflicts and collision types were not measured separately.

2. Training–evaluation divergence: The gap between training- and evaluation-reward changes is more pronounced in the reported dual-agent runs, with evaluation-reward differences of 40–158% despite lower training rewards in two algorithms. These descriptive differences do not establish a generalization effect.

3. Algorithm-dependent pattern: MADDPG shows positive descriptive reward differences in both settings, but the underlying mechanism and reliability of this pattern were not tested separately.

### 8.5.2 Computational Overhead

The training-time overhead for A* integration in multi-agent scenarios ranges from 33% to 66%, as reported by algorithm in Table 8.8.

Table 8.8: Training Time Overhead (2 Agents)

| **Algorithm** | **Base Time** | **A* Time** | **Overhead** |
|---|---|---|---|
| MAPPO | 2h 9m | 3h 14m | +50% |
| MADDPG | 2h 10m | 2h 53m | +33% |
| QMIX | 2h 15m | 3h 44m | +66% |

### 8.5.3 Implications for Future Multi-Agent Path Planning

The results suggest that the effectiveness of heuristic guidance in cooperative environments depends critically on the level at which coordination is enforced. Independent A* planning, while effective in single-agent navigation, does not inherently guarantee joint feasibility in multi-agent systems. Future work should therefore reconsider how planning and learning components interact under coordination constraints.

1. Conflict-aware path planning: Extending classical A* into a space-time formulation, where agents are modeled as dynamic obstacles, could prevent both spatial overlap and temporal conflicts. Methods such as Cooperative A* or reservation-based planning explicitly encode trajectory-level coordination, enabling joint feasibility during path computation.

2. Joint or priority-constrained optimization: Incorporating priority rules or partial joint cost formulations can mitigate bottleneck competition when multiple agents converge toward shared corridors. Such mechanisms shift optimization from purely individual shortest paths toward coordination-aware navigation policies, though potentially at increased computational cost.

3. Learning-level coordination incentives: Beyond planner modification, embedding inter-agent proximity penalties directly into the reward function may encourage

proactive avoidance behaviors. This approach aligns the learning objective with coordination requirements, complementing heuristic guidance without fully centralizing planning.

Collectively, these directions highlight that hybrid planning–learning architectures must balance individual optimality, collective feasibility, and scalability considerations when extended to cooperative multi-agent domains.

## 8.6 Summary

Table 8.9 summarizes the outcomes of the three hypotheses evaluated in this chapter.

Table 8.9: Hypothesis Validation Summary

| Hypothesis | Core Statement | Result |
|---|---|---|
| H1 | Lower collision-penalty proxy maintained in multi-agent | **Not Supported** |
| H2 | Multi-agent challenges diminish A* effectiveness | Partially Supported |
| H3 | Eval improvement exceeds training improvement | **Strongly Supported** |

This experiment extends the investigation of A* integration to genuine multi-agent scenarios, revealing several key findings:

1. A* guidance does not maintain lower collision-penalty proxy values in the two-agent scenarios (+0.5% average vs. -17.6% in the single-agent scenarios). Because collision types and path conflicts were not measured separately, conflicts between independently planned paths remain a plausible explanation rather than a confirmed cause.

2. The training–evaluation gap becomes more pronounced: all algorithms show larger descriptive evaluation-reward changes (40–158%) than training-reward changes (-24% to +6%). Because generalization was not measured separately, this pattern does not establish a generalization effect.

3. MADDPG shows positive descriptive reward differences in the reported single- and dual-agent comparisons. Deterministic continuous control is a possible explanation, but its contribution was not tested separately.

4. QMIX is the only tested algorithm with a lower collision-penalty proxy under A* in the multi-agent setting. Because this proxy combines agent–obstacle and agent–agent collision penalties, this result does not identify which collision type changed or establish a coordination mechanism.

These findings reinforce the conclusion that heuristic guidance alone does not guarantee coordination in cooperative environments. Effective multi-agent integration likely requires either planner-level conflict awareness or learning-level mechanisms that explicitly model inter-agent interactions.

# Chapter 9

# Evaluation of Task-Adaptive Waypoint-Aware Transformer Policies in Dual-Agent Navigation

## 9.1 Introduction

The results presented in Chapter 7 suggest that TA-WAT may achieve higher observed final-performance values than both the Fully Connected MLP Policy Baseline and the Standard Transformer. In the reported single-agent comparison, TA-WAT produced a final-window evaluation-reward value 40.3% higher than that of the Standard Transformer. However, given the limited statistical evidence, these findings should be interpreted as indicative rather than conclusive. A critical question therefore remains: Do these descriptive differences persist when the number of agents increases?

This chapter examines architectural performance when moving from single-agent to multi-agent navigation. The environment and training settings are kept the same to isolate the effect of adding a second agent. The focus is on basic pathfinding, without modeling explicit cooperation or competition. The multi-agent setup is used to study the added complexity from interaction and collision avoidance. Because multi-agent tasks are more complex, training time sufficient for single-agent convergence is not enough for dual-agent cases. Therefore, this chapter focuses on learning dynamics and convergence trends rather than final performance.

## 9.2 Experimental Design

### 9.2.1 Research Hypotheses

Building on the findings from Chapter 7, this experiment tests two hypotheses regarding multi-agent scaling:

- **H1**: Multi-agent scenarios require a longer observed training horizon to reach the convergence criterion than the corresponding single-agent settings.
- **H2**: TA-WAT maintains its descriptive advantage over the Standard Transformer in multi-agent scenarios, although the magnitude of the difference may vary.

These hypotheses aim to bridge single-agent findings to multi-agent deployment, providing practical guidance for architecture and algorithm selection in real-world multi-robot systems.

### 9.2.2 Experimental Setup

The experimental configuration directly extends Chapter 7 with a single modification: the number of agents is increased from one to two. All other parameters including world size, obstacle density, sensor resolution, A* waypoint generation, and training hyperparameters remain identical. Table 9.1 summarizes this controlled configuration comparison. The observed differences are therefore associated with the increase in agent count under the tested setup.

Table 9.1: Single-Agent vs. Multi-Agent Configuration Comparison

| **Parameter** | **Chapter 7 (Single)** | **Chapter 9 (Dual)** |
|---|---|---|
| Number of Agents | 1 | 2 |
| Random Seed (reported training-iteration-500 runs) | – | 96 |
| Algorithms Tested | MAPPO, MADDPG, QMIX | MAPPO, MADDPG, QMIX |
| Architectures | Standard Transformer, TA-WAT | Standard Transformer, TA-WAT |

The dual-agent scenario introduces several new challenges not present in single-agent navigation:

1. **Dynamic Obstacle Avoidance**: Each agent must treat the other agent as a moving obstacle, requiring real-time trajectory prediction and adaptation. Unlike static obstacles, other agents have unpredictable behaviors that evolve during training.
2. **Goal Interference**: Agents may have conflicting optimal paths, necessitating implicit or explicit coordination. The A* waypoints are computed independently for each agent, potentially creating path conflicts.

3. **Credit Assignment Complexity**: Shared rewards and penalties make it more difficult to attribute outcomes to individual agent decisions. A collision between agents affects both agents' rewards, complicating learning signals.

4. **Observation Space Expansion**: Each agent's observation must include information about other agents' positions and velocities, increasing input dimensionality and requiring the network to process inter-agent relationships.

### 9.2.3 Convergence Considerations

All analyses in this chapter are based on the training and evaluation curves over 500 training iterations. The extended training horizon allows a clearer observation of the long-term learning dynamics and convergence behavior of the compared algorithms. The results indicate that MADDPG and QMIX continue to improve and maintain relatively stable performance, while MAPPO shows a clear performance degradation during the later stages of training. Therefore, the conclusions in this chapter are drawn from the extended 500-iteration training experiments.

## 9.3 Results

### 9.3.1 Learning Dynamics: Average Episode Reward

Figure 9.1 illustrates the training progression of average episode reward across all six configurations over 500 training iterations.

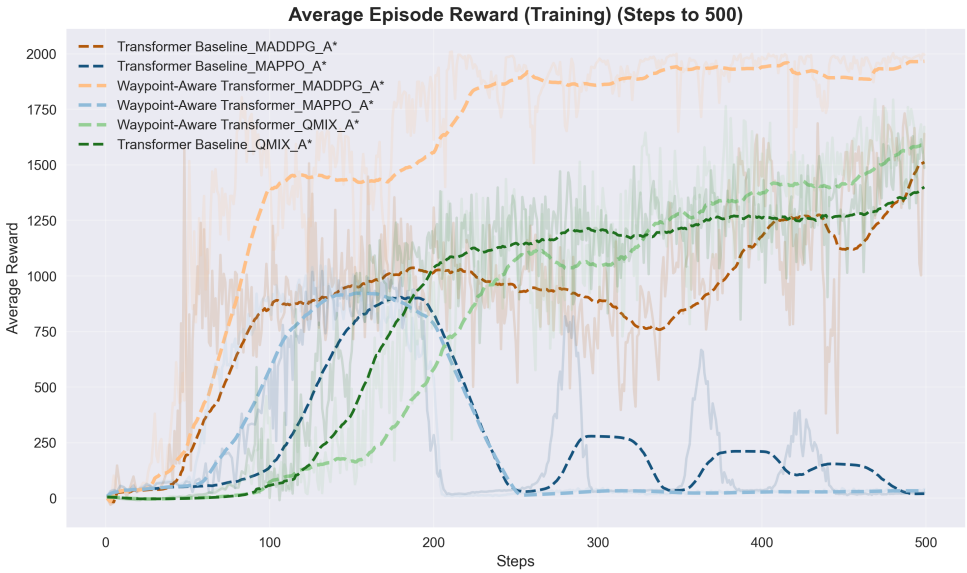



Figure 9.1: Average Episode Reward during training (iterations 1–500).

**Relative Performance:** At training iteration 500, MADDPG with TA-WAT achieves the strongest observed training return (final reward ≈ 1930), while both QMIX variants show sustained late-stage growth (final rewards ≈ 1335 and ≈ 1505). In contrast, both MAPPO variants plateau early and then degrade in later stages.

**Learning Dynamics:** MADDPG shows rapid early gains and remains high-performing with moderate oscillations. QMIX shows delayed but robust long-run improvement. MAPPO exhibits unstable late-stage behavior, including sharp drops in both training and evaluation proxies.

**Convergence Assessment:** By training iteration 500, MADDPG and QMIX are usable for final comparison, while MAPPO still degrades.

### 9.3.2 Evaluation Reward Analysis

Figure 9.2 presents the evaluation reward progression over training.

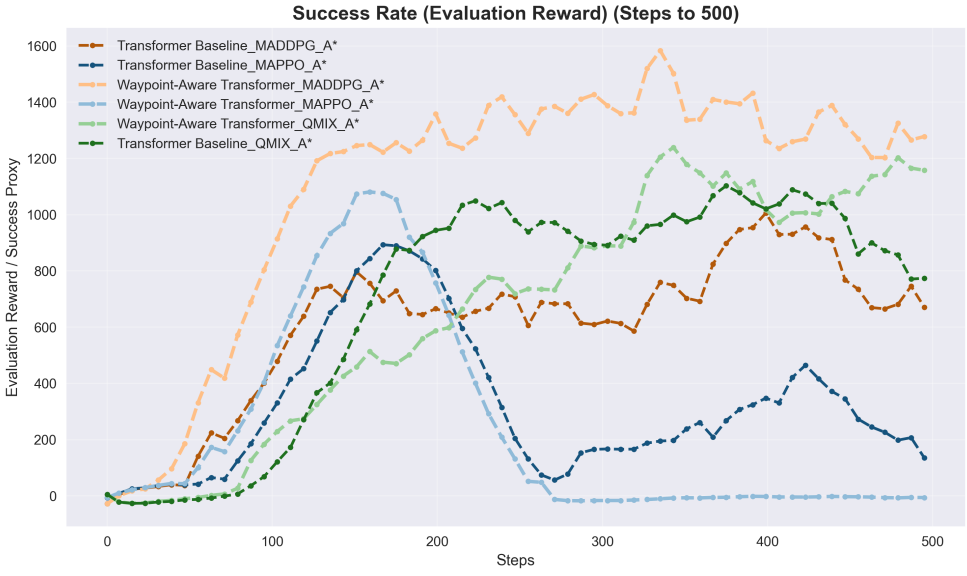


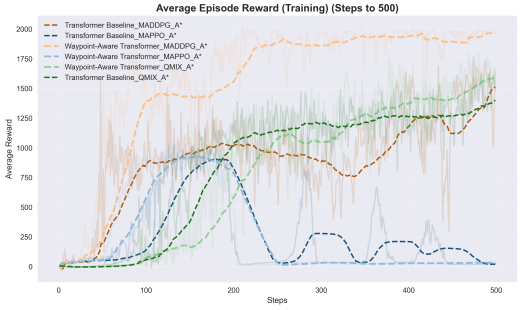

Figure 9.2: Evaluation reward over training iterations. This is a reward value, not a binary success rate. Evaluation is performed with A*-based reward shaping disabled. For Standard Transformer variants, this corresponds to evaluation without A*-derived waypoint input. For TA-WAT variants, the policy still receives waypoint observation features, so the evaluation measures performance without reward-level A* guidance but not without all A*-derived information.

The evaluation-reward trajectories follow the training-reward trends but reveal sharper algorithm divergence at training iteration 500. MADDPG with TA-WAT remains highest overall (final evaluation reward ≈ 1278), QMIX with TA-WAT shows strong late-stage gains (final evaluation reward ≈ 1157), and both MAPPO variants deteriorate substantially in late-stage evaluation (MAPPO with TA-WAT approaches

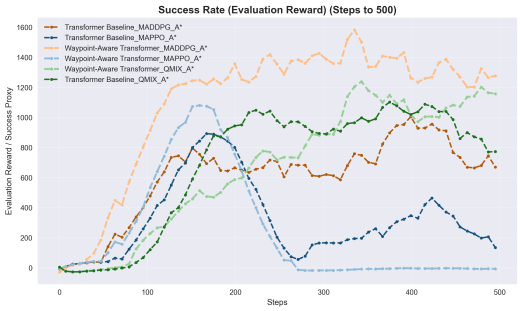
zero or becomes negative).

### 9.3.3 Learning Dynamics Analysis

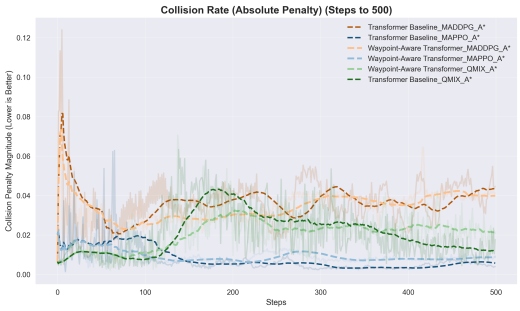
Figure 9.3 shows training progress over 500 training iterations.



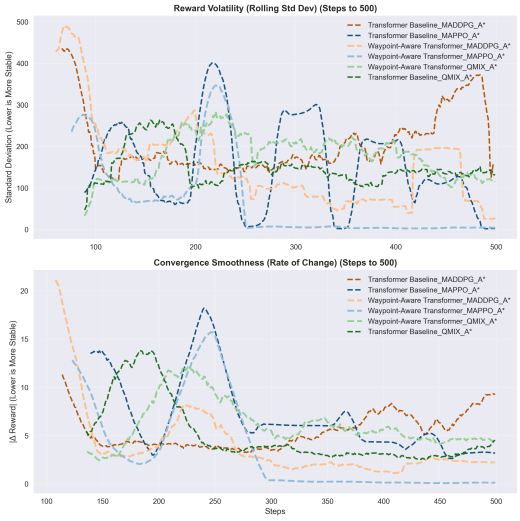

(a) Average Episode Reward




(b) Evaluation Reward (legacy embedded plot label: "Success Rate")




(c) Collision-Penalty Proxy




(d) Convergence Stability

Figure 9.3: Training dynamics comparison of different algorithms for two-agent navigation. Evaluation disables A*-based reward shaping; Waypoint-Aware Transformer variants may still receive waypoint observation features.

The QMIX collision pattern highlights a critical observation: reward improvement does not necessarily imply safety improvement. Even under a 500-iteration horizon, reward gains may coexist with collision-proxy trade-offs. This emphasizes the necessity of multi-objective monitoring in multi-agent learning.

### 9.3.4 Convergence Stability Analysis

Figure 9.4 presents two complementary stability metrics: reward volatility (rolling standard deviation) and convergence smoothness (rate of change).

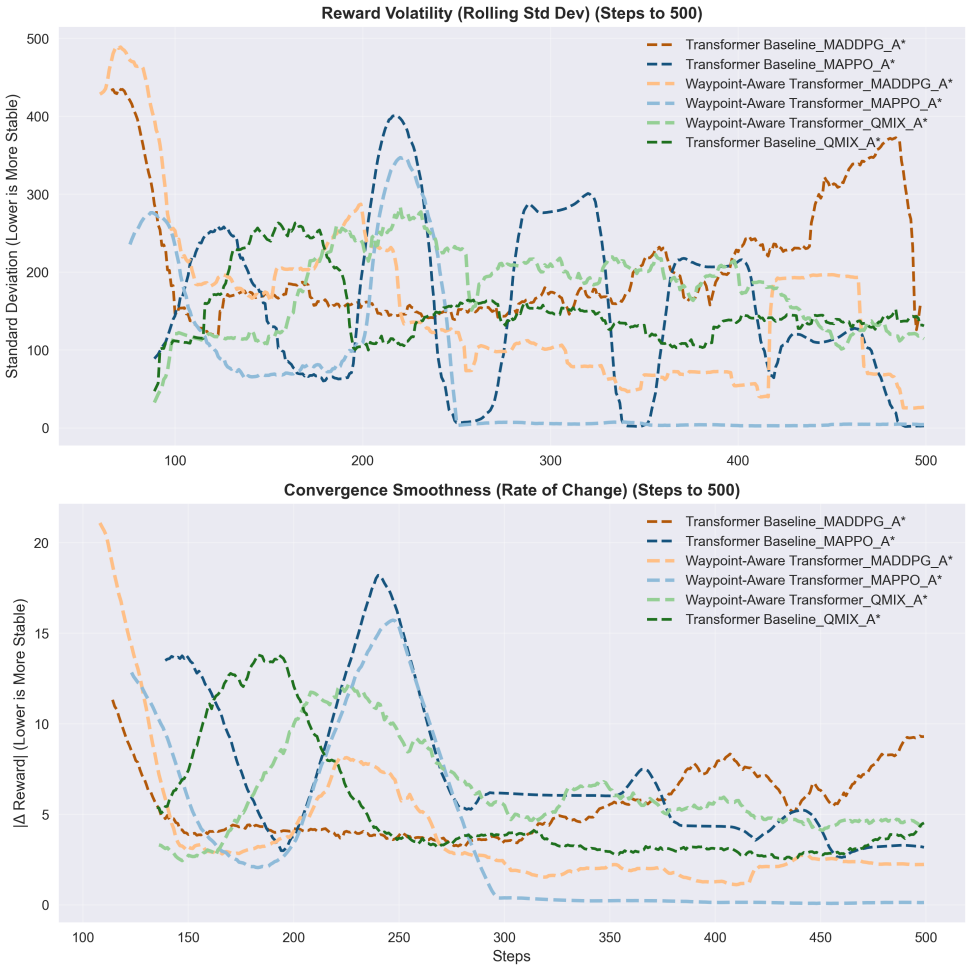



Figure 9.4: Top: Reward Volatility (Rolling Standard Deviation) lower values indicate more stable training. Bottom: Convergence Smoothness (Rate of Change) lower values indicate smoother learning progression.

**Reward Volatility Analysis (Top Panel):** Over the full 500-iteration horizon, MAPPO's late-stage low volatility coincides with a collapsed evaluation-return trajectory, indicating convergence to poor policies rather than high-quality stability. QMIX maintains moderate volatility with improving long-horizon performance. MADDPG remains high-performing but shows comparatively larger oscillations, especially with the Standard Transformer Policy Baseline.

**Convergence Smoothness Analysis (Bottom Panel):** The rate-of-change metric ($|\Delta\text{Reward}|$) quantifies policy update smoothness. All methods exhibit elevated

change rates during active learning (training iterations 60–100). In the full 500-iteration view, MAPPO's low late-stage variation corresponds to policy collapse in evaluation return rather than robust convergence. In contrast, QMIX keeps moderate updates with sustained long-horizon gains, and MADDPG maintains high performance with comparatively larger variance.

### 9.3.5 Quantitative Performance Summary

Table 9.2 summarizes the performance of different algorithm–architecture combinations in the dual-agent navigation task at the 500-iteration training horizon. At training iteration 500, the MADDPG waypoint-aware configuration showed higher training- and evaluation-reward values and a lower coefficient of variation than the corresponding Standard Transformer configuration. These are descriptive results from the tested runs and do not establish a statistically significant effect or identify the contribution of an individual architectural component.

Table 9.2: Dual-Agent Navigation Performance Comparison at Training Iteration 500 across Transformer-Based Architectures and DRL Algorithms

| Architecture | Algorithm | Avg Reward | Evaluation Reward | Collision-Penalty Proxy | CV | Training Time | Status |
|---|---|---|---|---|---|---|---|
| Standard Transformer | MADDPG | 978.33 | 670.93 | 0.0360 | 0.361 | 19h 49m | Converged |
| TA-WAT | MADDPG | **1644.32** | **1277.83** | 0.0340 | **0.297** | 15h 03m | Converged |
| Standard Transformer | MAPPO | 275.57 | 135.05 | **0.0074** | 1.204 | 12h 04m | Training Collapse |
| TA-WAT | MAPPO | 264.00 | -5.99 | 0.0090 | 1.375 | 19h 13m | Training Collapse |
| Standard Transformer | QMIX | **984.82** | 774.08 | **0.0210** | **0.475** | 14h 40m | Still Improving |
| TA-WAT | QMIX | 946.49 | **1157.42** | 0.0212 | 0.591 | 14h 47m | Still Improving |

Among the evaluated approaches, MADDPG demonstrates the strongest observed overall performance. In particular, TA-WAT has the highest average reward (1644.32) and evaluation reward (1277.83), while maintaining a relatively low collision-penalty proxy value (0.0340). In addition, it exhibits the lowest coefficient of variation (CV = 0.297) in these runs. These descriptive results suggest that the waypoint-input condition may be useful with MADDPG in the tested dual-agent environment; no multi-seed significance test was conducted for this comparison.

MAPPO shows a clear late-stage collapse in the 500-iteration run, marked by extreme gradient spikes and aggressive policy updates, leading to sharp drops in evaluation returns. This behavior indicates that the degradation is mainly caused by optimization instability, with possible minor overfitting, rather than transient noise or under-training. A thorough investigation into the causes and mitigation of MAPPO's late-stage instability will be addressed in future work.

QMIX demonstrates moderate but steadily improving performance. Both QMIX configurations achieve relatively high evaluation-reward values (774.08 and 1157.42) while maintaining low collision-penalty proxy values (approximately 0.021). However, their reward performance remains slightly lower than that of the Waypoint-Aware

Transformer MADDPG model. The observed learning trends suggest that QMIX may not have fully converged within the 500-iteration training horizon and could potentially achieve further improvements with extended training.

Overall, MADDPG combined with TA-WAT shows the strongest observed balance among reward performance, variability, and training time in the reported runs. Meanwhile, MAPPO appears sensitive to the current reward shaping and training dynamics, whereas QMIX shows mixed but still developing performance characteristics. These comparisons are descriptive and were not supported by a multi-seed significance test.

## 9.4 TA-WAT vs. Standard Transformer

A central question of this experiment is whether the descriptive differences observed for TA-WAT persist in multi-agent scenarios. This section provides a comprehensive comparison.

### 9.4.1 Overall Optimization Effect

Table 9.3 presents the aggregated optimization effects across all algorithms.

Table 9.3: TA-WAT vs. Standard Transformer: Aggregated Comparison

| Metric | Standard Transformer | TA-WAT | Change |
|---|---|---|---|
| Average Reward (mean) | 746.24 | 951.60 | **+27.5%** (better) |
| Evaluation Reward (mean) | 526.69 | 809.75 | **+53.8%** (higher) |
| Collision-Penalty Proxy (mean) | 0.0215 | 0.0214 | **-0.5%** (lower) |
| Stability CV (mean) | 0.680 | 0.754 | +10.9% (less stable) |

The aggregated training-iteration-500 values are descriptively higher for the waypoint-aware configurations in training and evaluation reward, while stability is worse and the effect varies across algorithms. On average, training reward increases by 27.5% and evaluation reward by 53.8%; these values describe the observed runs and do not establish a statistically significant architecture effect. The collision-penalty proxy is nearly unchanged, with a 0.5% lower value. This proxy combines agent–obstacle and agent–agent collision penalties and is not a direct collision rate; the result therefore cannot establish that collision frequency or safety risk is unchanged. The coefficient of variation increases by 10.9%, indicating a modest decline in training stability and a possible performance–stability trade-off.

### 9.4.2 Algorithm-Specific Optimization Effects

The optimization effect varies substantially across algorithms, as shown in Table 9.4.

In the tested MADDPG runs, TA-WAT showed a 68.1% higher average-reward value, a 90.5% higher evaluation-reward value, a 5.6% lower collision-penalty proxy,

Table 9.4: Performance differences (Δ) between TA-WAT and the Standard Transformer at training iteration 500.

| Algorithm | Reward | Evaluation Reward | Collision-Penalty Proxy | CV | Net Effect | Interpretation |
|---|---|---|---|---|---|---|
| MADDPG | +68.1% | +90.5% | -5.6% ↓ | -17.7% (better) | **Strongly Positive** | Large performance and stability gains |
| MAPPO | – | – | – | – | – | Instability and policy collapse risk |
| QMIX | -3.9% | +49.5% | +1.0% ↑ | +24.4% (worse) | Mixed | Higher evaluation reward but reduced stability |

and a 17.7% lower coefficient of variation than the Standard Transformer. Because the configurations were not subjected to component-level ablation or multi-seed significance testing in this dual-agent experiment, these descriptive differences cannot be attributed to enhanced representational capacity, relational reasoning, or a waypoint-aware attention mechanism. The larger difference observed for MADDPG may motivate further investigation of interactions between waypoint inputs and continuous control, but this mechanism was not tested.

Due to pronounced instability observed in MAPPO during later training stages, its results are excluded from detailed quantitative analysis to avoid potentially misleading interpretations. This behaviour may indicate sensitivity under the tested configuration, but the experiment does not isolate whether the pattern is related to on-policy optimization, the observation encoding, multi-agent non-stationarity, sparse rewards, or their interaction.

For QMIX, the descriptive results are mixed. While the evaluation-reward value is 49.5% higher, the average-reward value is 3.9% lower, and both the collision-penalty proxy and CV are modestly higher. These results do not establish that TA-WAT improves task completion or introduces variance in value estimation. A possible explanation involves an interaction between waypoint inputs, the discrete action space, and the value-decomposition structure of QMIX, but this mechanism was not tested.

Overall, the observed differences between TA-WAT and the Standard Transformer are algorithm- and metric-dependent. MADDPG shows the strongest positive descriptive pattern, MAPPO exhibits unstable training dynamics under the current configuration, and QMIX shows mixed outcomes. Because the relevant mechanisms were not separately tested, the results do not establish whether these patterns arise from representation learning, optimization strategy, action-space structure, or their interaction.

## 9.5 Hypothesis Validation and Analysis

### 9.5.1 H1: Extended Training Requirements

**Result: Supported.**

The 500-iteration results indicate that multi-agent training requires a longer observed horizon than the single-agent tasks. Table 9.5 reports the convergence indicators used for this comparison.

Table 9.5: H1 Validation: Convergence Requirement Comparison (500-Iteration Evidence)

| Indicator | Single-Agent | Dual-Agent (This Chapter) |
|---|---|---|
| Converged within 100 training iterations | Yes | No |
| Practical convergence horizon | 75–94 | ≥400 (algorithm-dependent) |
| CV range (final) | 0.21–0.31 | 0.297–1.375 |
| Training-iteration-500 trend status | Stable | MADDPG/QMIX stable, MAPPO Collapse |

The single-agent experiment reached its convergence criterion within 75–94 training iterations and showed relatively low volatility. In contrast, the two-agent runs required a longer observed training horizon. Even at training iteration 500, the algorithms showed different trajectories: MADDPG and QMIX reached comparatively stable ranges, while MAPPO exhibited late-stage instability and collapse risk. These descriptive observations support H1, but they were not subjected to a separate significance test.

### 9.5.2 H2: Descriptive Performance of the Waypoint-Aware Transformer

**Result: Mixed descriptive evidence; not statistically confirmed.**

The evidence in Table 9.6 indicates that TA-WAT shows higher aggregate training- and evaluation-reward values in the reported training-iteration-500 comparison, but the pattern is algorithm- and metric-dependent and is not statistically confirmed.

Table 9.6: H2 Validation: Relative Performance Changes of TA-WAT vs. the Standard Transformer. Single-agent data from Tables 7.4–7.5; dual-agent data from Tables 9.3–9.4.

| Level | Metric | Single-Agent | Dual-Agent | Change Pattern |
|---|---|---|---|---|
| *Symmetric Aggregate (Three-Algorithm Mean)* | | | | |
| All algorithms | Reward | +6.1% | +27.5% | Amplified |
| All algorithms | Evaluation-Reward Change | +12.9% | +53.8% | Different reward scales |
| All algorithms | Coefficient of Variation (CV) | −9.1% | −10.9% | Unfavourable in both settings |
| *Algorithm-Specific Effects* | | | | |
| MAPPO | Reward | +10.9% | −− | Not reported due to instability |
| MAPPO | Evaluation-Reward Change | +40.3% | −− | Not reported due to instability |
| MAPPO | Coefficient of Variation (CV) | −46.4% | −− | Not reported due to instability |
| QMIX | Reward | +1.1% | −3.9% | Reduced |
| QMIX | Evaluation-Reward Change | −5.1% | +49.5% | Different reward scales |
| QMIX | Coefficient of Variation (CV) | +0.5% | −24.4% | Reversed |
| MADDPG | Reward | +7.5% | +68.1% | Amplified |
| MADDPG | Evaluation-Reward Change | +7.7% | +90.5% | Different reward scales |
| MADDPG | Coefficient of Variation (CV) | +18.6% | +17.7% | Favourable in both settings |

Table 9.6 uses the same performance-direction convention as Table 7.7: positive values are favourable and negative values are unfavourable. For CV, a reduction is therefore reported as a positive stability change, whereas an increase is reported as a negative stability change.

The task-performance rows in both settings refer to relative changes in evaluation reward. However, their numerical magnitudes should not be compared as if they were on a common reward scale because the agent count, reward aggregation, and experimental configurations differ. At the symmetric aggregate level, TA-WAT has higher observed evaluation-reward values in both settings, but these descriptive results should not be interpreted as statistically confirmed benefits. Stability also changes unfavourably in the dual-agent setting, indicating that the architectural effect is conditional and metric-dependent.

At the algorithm level, MAPPO maintains a descriptive reward difference, but dual-agent data is not reported due to instability. In the dual-agent runs, QMIX shows a 49.5% higher evaluation-reward value, while MADDPG shows 68.1% higher training reward and a 90.5% higher evaluation-reward value, with 17.7% lower variability. These values are descriptive rather than statistically confirmed effects.

Overall, TA-WAT shows a positive descriptive difference in some aggregate performance metrics, but the pattern varies across algorithms and metrics under multi-agent interaction. Because coordination mechanisms were not measured separately, the results do not establish that multi-agent interaction redistributes or causes these differences.

## 9.6 Discussion

### 9.6.1 Waypoint-Input Differences Across Algorithms and Agent Settings

Although MAPPO was unstable in this validation, QMIX and MADDPG show positive descriptive differences in selected metrics under the waypoint-input condition. The differences are conditional rather than uniform, and they do not establish the effect of any individual Transformer component or a consistent benefit across algorithms and agent counts. This is consistent with the descriptive differences reported in the single-agent setting (Chapter 7) and the dual-agent study under interaction constraints (Chapter 8). However, the strength and direction of the differences vary across algorithms and metrics. Therefore, while TA-WAT shows promise, further systematic evaluation is required to determine its general effectiveness and to test how interaction dynamics and algorithm-specific updates may relate to its performance.

### 9.6.2 MAPPO Failure Mode: Preferred and Alternative Explanations

Based on current evidence, the most plausible explanation for MAPPO degradation remains *instability-dominant optimization failure*: downturn signals emerge near

the end of the 200-iteration window and intensify by the 500-iteration horizon, accompanied by gradient spikes and aggressive updates, making random fluctuations unlikely. While partial overfitting may contribute, the primary driver appears to be optimization instability. A possible underlying factor is a high learning rate, which can amplify gradient oscillations and destabilize training. To ensure experimental consistency, hyperparameters were kept fixed across all scenarios; however, single-agent and multi-agent settings may differ in their sensitivity to these parameters, affecting the algorithm's stability. Future work should systematically explore the interaction between learning rates, agent count, and update dynamics to better understand and mitigate MAPPO collapse.

## 9.7 Summary

Table 9.7 summarizes the outcomes of the hypotheses evaluated in this chapter.

Table 9.7: Hypothesis Validation Summary

| Hypothesis | Core Statement | Result |
|---|---|---|
| H1 | Multi-agent requires longer training | **Supported** |
| H2 | Waypoint-Aware Transformer maintains advantage | **Supported with Structural Moderation** |

This chapter investigated the scaling behavior of Transformer architectures when transitioning from single-agent to dual-agent navigation scenarios. The learning dynamics reveal important insights with practical implications:

1. **Waypoint-aware configurations show higher descriptive aggregate training- and evaluation-reward values at training iteration 500, but the pattern is not uniform across algorithms and is not statistically confirmed.** The dual-agent average-reward difference is +27.5%, and the evaluation-reward difference is +53.8%. The collision-penalty proxy is nearly unchanged at training iteration 500 (-0.5%), while the raw aggregate CV increases by 10.9%, equivalent to a $-10.9\%$ stability-performance change under the convention used in Table 9.6. Single-agent and dual-agent values are both evaluation-reward measures, but their scales are not directly comparable because the experimental and reward-aggregation settings differ.

2. **The descriptive differences vary across algorithms.** In the dual-agent runs, MADDPG has a +68.1% training-reward difference, a +90.5% evaluation-reward difference, and a $-17.7\%$ CV difference. QMIX has a $-3.9\%$ training-reward difference, a +49.5% evaluation-reward difference, and a +24.4% CV difference. MAPPO cannot sustain stable long-horizon behavior in this setting.

These findings provide preliminary descriptive evidence that some single-agent patterns may also appear in dual-agent experiments. However, the current results do not establish reliable transfer of architectural benefits or the contribution of any individual Transformer component.

# Chapter 10

# Discussion, Contributions, and Conclusion

## 10.1 Summary of Findings

This thesis builds upon the A*-guided Deep Reinforcement Learning framework proposed by Lopez-Yepez et al., and extends it through a systematic and reproducible investigation of planning-guided learning. It investigates two forms of A*-derived guidance. In the MLP baseline and Standard Transformer configurations, A* is used during training for waypoint-based reward shaping and is removed during testing, so the learned policy operates without A*-derived waypoint input. In the Task-Adaptive Waypoint-Aware Transformer configurations, A*-based reward shaping is disabled during testing, but the policy still receives 5-dimensional waypoint observation features. Therefore, these configurations should be interpreted as removing reward-level A* guidance during evaluation, but not all A*-derived information. The work further emphasizes reproducibility, architectural clarity, and algorithmic generality across controlled navigation settings.

### 10.1.1 A* Guidance Effects (Single-Agent)

1. **Lower collision-penalty proxy across algorithms.** In the single-agent scenarios (Chapter 6), A* guidance is associated with a 17.6% lower collision-penalty proxy on average across the tested algorithms (MAPPO: $-18.8\%$, MADDPG: $-12.5\%$, QMIX: $-21.7\%$). Negative values indicate lower absolute collision-penalty magnitude relative to the baseline. This proxy combines agent–obstacle and agent–agent collision penalties and is not a direct collision rate.

2. **Batch size sensitivity in on-policy learning.** MAPPO shows descriptive sensitivity to batch size when combined with A* guidance. Reduced sample diversity

and higher gradient variance are plausible explanations, but these mechanisms were not measured separately.

3. **Positive descriptive pattern for MADDPG.** MADDPG shows the largest positive descriptive differences under A* guidance in this comparison. Experience replay, continuous actions, and deterministic policies are possible explanations, but their individual effects were not tested.

4. **Computational overhead is substantial.** A* integration increases training time by 33%–97% depending on the algorithm (MAPPO: +97%, MADDPG: +45%, QMIX: +33%), where positive values indicate increased computational cost (i.e., worse efficiency). This overhead must be weighed against the observed performance differences in practical applications.

### 10.1.2 A* Guidance in Multi-Agent Scenarios

Chapter 8 extended the A* guidance investigation to genuine dual-agent navigation, revealing fundamental changes in how A* affects learning:

1. **Collision-penalty proxy reversal.** Contrary to the single-agent findings, A* guidance does not maintain lower collision-penalty proxy values in the dual-agent scenarios. The average proxy change reverses from −17.6% (single-agent) to +0.5% (dual-agent). Only QMIX has a lower proxy (−12.9%), while MAPPO (+2.9%) and MADDPG (+16.1%) have higher values. Because collision types and path conflicts were not measured separately, independently planned path conflicts remain a plausible explanation rather than a confirmed cause.

2. **Training-evaluation divergence.** The reported evaluation-reward differences (40–158%) are larger than the training-reward differences (−24% to +6%). QMIX shows the largest gap: −13.2% training reward and +158.0% evaluation reward. Because no separate generalization measure or powered multi-seed test was used, this divergence should not be described as confirmed improvement in policy generalization.

3. **MADDPG consistency across scales.** MADDPG shows the strongest positive descriptive pattern in this comparison, with +6.1% training reward and +139.5% evaluation reward relative to the baseline. Deterministic policies and experience replay are plausible explanations, but these mechanisms were not separately tested.

### 10.1.3 Transformer Architecture Effects

1. **The Standard Transformer shows lower descriptive values than the MLP baseline.** The Standard Transformer has a 19.3% lower average-reward value and a 20.3% lower final-window evaluation-reward value than the MLP baseline under the tested configuration. This comparison does not establish that model complexity or any individual Transformer component caused the difference. A similar descriptive pattern was observed in the Experimental Development and Design Decisions chapter.

2. **The complete waypoint-aware configuration shows positive descriptive results in some comparisons.** The Standard Transformer and TA-WAT use the same input dimension, network structure, relative positional encoding, gated FFN, local attention, and training settings. Their only intentional model-input difference is the shared five-slot waypoint field: it is zero-filled for the Standard Transformer and contains A*-derived values $(\Delta x, \Delta y, d, dx, dy)$ for TA-WAT. TA-WAT achieved the highest observed final-window evaluation reward in the MAPPO comparison (87.07 versus 77.87 reward units for the MLP baseline), corresponding to an observed relative difference of 11.8%. However, the five-seed analysis did not establish a statistically significant advantage. The comparison therefore does not establish separate effects for the shared Transformer components; the observed differences are associated with the complete waypoint-input condition and require further validation.

3. **Convergence iteration vs. final performance trade-off.** The MLP baseline reaches its convergence criterion 5.2× earlier than TA-WAT (18 versus 94 training iterations) but has a lower observed final-performance value. Because the final-performance difference was not statistically confirmed, this provides only tentative guidance for architecture selection.

### 10.1.4 Multi-Agent Scaling Effects

Chapter 9 extended the investigation to dual-agent navigation, keeping all other parameters identical to the single-agent experiments. The key findings are:

1. **Substantially increased training requirements.** All six dual-agent configurations remain in active learning at training iteration 200, and practical convergence is estimated at 400+ training iterations, at least 2–3× the single-agent requirement. At training iteration 500, MADDPG/QMIX show usable long-horizon trends.

2. **Higher aggregate descriptive reward values.** Compared with the Standard Transformer Policy Baseline at training iteration 500, the Waypoint-Aware Transformer has a +27.5% average-reward difference and a +53.8% evaluation-reward difference, while the collision-penalty proxy is nearly unchanged (−0.5%).

3. **Stability does not transfer uniformly.** Although the aggregate reward values are higher, aggregate stability is weaker in dual-agent training (CV +10.9% under Task-Adaptive Waypoint-Aware Transformer). Thus, the descriptive reward and stability measures do not change in the same direction.

4. **Algorithm-dependent descriptive differences.** MADDPG has a +68.1% reward difference, a +90.5% evaluation-reward difference, and a −17.7% CV difference. QMIX has a −3.9% reward difference, a +49.5% evaluation-reward difference, and a +24.4% CV difference. MAPPO shows instability-dominant collapse risk in long-horizon training.

### 10.1.5 Integrated Findings

Table 10.1 summarizes the key experimental results across all single-agent and dual-agent configurations:

Table 10.1: Summary of Key Performance Metrics Across Experiments

| Configuration | Avg Reward | Final-Window Evaluation Reward | Collision-Penalty Proxy | Convergence Iteration or Status | Training Time |
|---|---|---|---|---|---|
| *Single-Agent: Algorithm Comparison (Chapter 6)* | | | | | |
| MADDPG-A* | 106.94 | 80.68 | 0.0049 | 22 | 3h 8m |
| MAPPO-A* | 97.45 | 77.87 | 0.0026 | 18 | 2h 22m |
| QMIX-A* | 100.63 | 68.19 | 0.0047 | 65 | 2h 47m |
| *Single-Agent: Architecture Enhancement (Chapter 7)* | | | | | |
| MAPPO-MLP-A* | 97.45 | 77.87 | 0.0026 | 18 | 2h 23m |
| MAPPO-Waypoint-Aware Transformer-A* | 87.27 | 87.07 | 0.0126 | 94 | 2h 13m |
| MADDPG-Waypoint-Aware Transformer-A* | 100.02 | 64.27 | 0.0121 | 65 | 4h 5m |
| QMIX-Waypoint-Aware Transformer-A* | 100.61 | 73.21 | 0.0045 | 61 | 4h 3m |
| *Dual-Agent: A* Feasibility (Chapter 8)* | | | | | |
| MADDPG-A* (2 agents) | 1596.06 | 1014.71 | 0.0180 | 131 | 2h 53m |
| MADDPG-Base (2 agents) | 1504.23 | 423.62 | 0.0155 | 85 | 2h 10m |
| MAPPO-A* (2 agents) | 1440.35 | 938.31 | 0.0215 | 38 | 3h 14m |
| MAPPO-Base (2 agents) | 1902.80 | 668.16 | 0.0209 | 61 | 2h 9m |
| QMIX-A* (2 agents) | 1373.96 | 834.04 | 0.0176 | 126 | 3h 44m |
| QMIX-Base (2 agents) | 1583.14 | 323.27 | 0.0202 | 114 | 2h 15m |
| *Dual-Agent: Multi-Agent Scaling (Chapter 9, training iteration 500)* | | | | | |
| MADDPG-Standard Transformer | 978.33 | 670.93 | 0.0360 | Converged | 19h 49m |
| MADDPG-TA-WAT | 1644.32 | 1277.83 | 0.0340 | Converged | 15h 03m |
| MAPPO-Standard Transformer | 275.57 | 135.05 | 0.0074 | Collapse risk | 12h 04m |
| MAPPO-TA-WAT | 264.00 | -5.99 | 0.0090 | Collapse risk | 19h 13m |
| QMIX-Standard Transformer | 984.82 | 774.08 | 0.0210 | Still improving | 14h 40m |
| QMIX-TA-WAT | 946.49 | 1157.42 | 0.0212 | Still improving | 14h 47m |

Task metric: final-window evaluation reward (reward units); cross-setting scales are not directly comparable.

*Note:* Both the single-agent and dual-agent task columns report final-window evaluation reward. Their scales are not directly comparable because the agent count, reward aggregation, and experimental configurations differ. The collision-penalty proxy combines agent–obstacle and agent–agent collision penalties and is not a direct collision rate.

## 10.2 Contributions

The main contributions of this thesis are summarized as follows:

1. **A reproducible experimental framework for A*-guided multi-agent reinforcement learning.** This thesis establishes a unified and systematically controlled experimental pipeline for integrating A*-guided reward shaping into deep reinforcement learning, enabling rigorous comparison across algorithms (MAPPO, MADDPG, QMIX) and architectures (MLP, Transformer) under controlled conditions. This framework addresses reproducibility and methodological ambiguities present in prior studies.

2. **Algorithm-dependent effectiveness analysis of planning-guided rewards.** A comparative empirical analysis reveals algorithm-dependent descriptive differences under A* guidance. In the single-agent comparisons, the collision-penalty proxy is 12.5%–21.7% lower across the tested algorithms, whereas relative changes in final-window evaluation reward range from $-0.8\%$ to $+6.6\%$ depending on the algorithm.

3. **Controlled evaluation of Transformer and waypoint-input policy configurations.** The term Task-Adaptive Waypoint-Aware Transformer was introduced in this thesis for the evaluated waypoint-input configuration and does not refer to a previously established named technique. The contribution lies in incorporating and evaluating A*-derived waypoint observations within the tested Transformer policy, rather than in proposing a fundamentally new Transformer architecture. In the reported MAPPO comparison, the Standard Transformer produced lower descriptive performance values than the MLP baseline, while the Task-Adaptive Waypoint-Aware Transformer produced an 11.8% higher observed final-window evaluation-reward value than the MLP. However, the five-seed tests did not establish a statistically significant performance advantage. The result should therefore be treated as a descriptive architectural comparison rather than as confirmed evidence of improvement.

4. **Systematic architectural study of structured policy representations.** This thesis provides a cross-algorithm evaluation of Transformer-based policies within an A*-guided learning framework. Positive descriptive differences were observed in selected configurations, but the five-seed tests did not establish statistically significant improvement or architecture-level generality. The observed performance differences were algorithm- and metric-dependent, and the statistical analysis did not establish a general performance advantage. The contribution is therefore the controlled cross-algorithm evaluation rather than proof of architecture-level superiority or generality.

5. **Quantification of computational trade-offs.** The computational overhead of A* guidance (33%–97% training time increase) and Transformer architectures is

explicitly quantified, providing practitioners with actionable guidance for architecture and algorithm selection in resource-constrained scenarios.

6. **Clarification of the reported A* guidance patterns.** The experimental results distinguish descriptive changes in final-window evaluation reward and the collision-penalty proxy from changes in convergence and training time. Because evaluation reward is not a binary success rate, the collision proxy is not a direct collision rate, and the mechanisms were not separately tested, these findings do not establish that A* guidance primarily improves navigation quality.

7. **Identification of an observed collision-metric reversal in multi-agent settings.** Chapter 8 reports that the collision-penalty proxy associated with independent A* path planning decreases in the single-agent comparisons ($-17.6\%$ on average) but increases slightly in the dual-agent comparisons ($+0.5\%$ on average). This proxy combines agent–obstacle and agent–agent collision penalties. Inter-agent path conflicts at shared bottlenecks are a plausible explanation, but collision types and explicit path conflicts were not measured separately. The evaluation-reward values are also 40–158% higher in the reported A*-guided runs despite lower training rewards; these are descriptive differences rather than a separately tested generalization effect.

8. **Multi-agent scaling analysis of the waypoint-input configuration.** This thesis extends the evaluation from single-agent to dual-agent navigation. At training iteration 500, TA-WAT has a 27.5% higher aggregate training-reward value and a 53.8% higher evaluation-reward value than the Standard Transformer, while its aggregate CV is 10.9% higher. These descriptive differences are algorithm- and metric-dependent and were not confirmed by a multi-seed significance test.

9. **Algorithm-dependent scaling characterization.** The per-algorithm analysis reports different descriptive patterns under multi-agent scaling: MADDPG has higher reward values and lower variability, QMIX has mixed reward and stability values, and MAPPO shows collapse risk in the reported long-horizon runs. These observations motivate further algorithm–architecture matching studies but do not establish a general scaling benefit.

## 10.3 Limitations

Several limitations of the current study should be acknowledged:

1. **Batch size interaction not exhaustively explored.** The interaction between

batch size and structured guidance was not systematically investigated. In Chapter 6, MAPPO with A* guidance showed reduced average reward compared to the baseline, potentially due to the smaller batch size (30,000 vs. 60,000 frames) reducing sample diversity from A*-structured trajectories. A comprehensive study of batch size effects across algorithms would provide deeper insights.

2. **Limited multi-agent scale.** Chapter 9 extended the investigation to dual-agent scenarios, partially addressing the single-agent limitation. However, only two agents were tested. The generalization of the observed stability reversal and algorithm-dependent scaling patterns to larger agent populations (3+ agents) remains to be validated.

3. **Long-horizon but still heterogeneous convergence in multi-agent experiments.** Extending training to 500 training iterations improved reliability versus the 200-iteration analysis, but algorithm behavior remains heterogeneous: MADDPG is converged, QMIX is still improving, and MAPPO shows instability/collapse risk. Therefore, some architecture-algorithm conclusions may still shift under even longer horizons.

4. **Independent path planning only.** Chapter 8 employed independent A* computation for each agent without inter-agent coordination. The observed collision reduction reversal is therefore specific to this planning strategy; conflict-aware alternatives (space-time A*, priority-based planning) were not tested and might restore collision reduction benefits in multi-agent settings.

5. **Environment complexity.** Experiments were conducted on "Easy" difficulty maps with relatively simple obstacle configurations. The effectiveness of A* guidance and Transformer architectures in more complex environments with dynamic obstacles or higher obstacle density requires further investigation.

6. **Transformer optimization scope.** The comparison did not independently isolate the effects of waypoint observations, relative positional encoding, gated FFN, or local attention. Although the Standard Transformer and waypoint-aware Transformer were intended to share the same core architecture, no component-level ablation was conducted. Consequently, the study cannot determine the individual contribution of these components to performance, stability, or robustness. The individual contribution of each optimization technique was not isolated, limiting understanding of which components are most critical for the observed stability reversal in multi-agent settings.

7. **Hyperparameter sensitivity.** The study used fixed hyperparameters across most experiments, with only batch size adjusted for Transformer stability. A

more comprehensive hyperparameter search might reveal configurations that better balance the trade-offs observed between architectures and algorithms, particularly for the MADDPG performance-stability trade-off in multi-agent settings.

8. **Unvalidated waypoint-threshold sensitivity.** The look-ahead waypoint mechanism used a fixed threshold of $d_{\min} = 0.5$ in all reported experiments. Nearest-node selection and look-ahead selection were not compared through a controlled multi-seed ablation, and the threshold was not varied systematically. Consequently, the study does not establish that the look-ahead mechanism improves performance or that $d_{\min} = 0.5$ is optimal.

9. **Navigation in constrained environments.** Agents can become trapped in L- and U-shaped obstacle formations, even with A*-based reward shaping. The local nature of the shaping signal and multi-agent coordination demands are plausible explanations, but these mechanisms were not measured separately. The tested method does not fully resolve navigation in highly constrained environments.

## 10.4 Future Work

Based on the findings and limitations of this study, several promising directions for future research are identified:

1. **Look-ahead waypoint ablation and threshold sensitivity.** Future work should compare nearest-node waypoint selection with the implemented look-ahead rule and evaluate multiple threshold values, for example $d_{\min} \in \{0.25, 0.5, 0.75, 1.0\}$, using identical maps, training budgets, algorithms, and random seeds. The comparison should examine evaluation reward, direct task success where available, collision-related measures, trajectory oscillation, path efficiency, and training stability. This would determine whether the look-ahead rule provides a reproducible benefit and whether its behaviour is sensitive to the selected threshold.

2. **Extended multi-agent training and larger teams.** Chapter 9 showed that 500 training iterations are necessary but not uniformly sufficient across algorithms. Future work should extend training horizons beyond 500 iterations to determine final QMIX convergence behavior and to evaluate whether MAPPO instability can be mitigated through improved optimization strategies. Additionally, scaling beyond two agents to teams of 3–10+ agents is required to assess whether the observed algorithm-dependent patterns remain consistent under increasing coordination complexity and expanded joint state spaces. To ensure rigorous evaluation, future studies should also incorporate statistical significance

testing, such as paired t-tests with Holm–Bonferroni correction, to determine whether observed performance differences across algorithms and architectures are statistically robust.

3. **Conflict-aware multi-agent path planning.** Chapter 8 showed that independent A* planning, while effective for single-agent navigation, reverses its collision reduction benefits in multi-agent settings due to bottleneck conflicts. Potential remedies include space-time A* formulations modeling agents as dynamic obstacles, priority- or joint-cost-based optimizations that promote coordination-aware trajectories, and learning-level incentives, such as inter-agent proximity penalties in the reward function, to encourage proactive avoidance without requiring fully centralized planning.

4. **Architectural ablation of Task-Adaptive Waypoint-Aware Transformer.** The mixed performance and stability outcomes observed in Chapter 9 suggest that different architectural components may contribute unevenly across algorithms. Future work should isolate the effects of relative positional encoding, gated feed-forward networks, and local attention mechanisms in order to clarify which components primarily enhance long-horizon performance and which primarily affect training stability.

5. **Adaptive batch size scheduling.** Dynamically adjusting batch size during training could allow algorithms to better accommodate the evolving influence of A* guidance. Early training might benefit from larger batches to stabilize learning from structured trajectories, while later stages could use smaller batches for fine-tuning. This may be particularly relevant for multi-agent settings, where the expanded joint observation space increases sample diversity requirements.

6. **Dynamic and complex environments.** Future work should evaluate A* guidance in more complex and dynamic environments, including settings with moving obstacles, time-varying goals, and higher obstacle densities. Such evaluations would provide insight into the robustness and generalization of the proposed approaches, particularly given that multi-agent interactions may amplify environmental complexity.

7. **Curriculum learning integration.** Combining A* guidance with curriculum learning strategies progressively adjusting environment difficulty, agent count, or A* guidance strength might improve both learning speed and final performance. The multi-agent findings suggest that gradually increasing agent count during training could help bridge the convergence gap between single-agent and multi-agent settings.

8. **Alternative path-planning algorithms.** Since the present experiments intentionally fixed A* as the planner, they do not establish whether the observed results are specific to A*. Future work should compare A* with alternative planners, including RRT, D*, and potential-field methods, to determine whether the observed patterns generalize to other forms of planning guidance.

9. **Trap problem mitigation.** Future work could explore strategies to address agents being trapped in L- and U-shaped obstacles. Optimizing reward shaping and tuning Transformer-related parameters may improve agents' ability to escape from constrained environments, enhancing learning stability and overall navigation performance.

10. **Scalability and mechanistic role of A*-derived guidance in TA-WAT for multi-agent navigation.** Building on the single-agent findings, future work should evaluate whether the performance and stability benefits of the **TA-WAT architecture** generalize to cooperative multi-agent settings with increasing coordination complexity, including dual-agent and larger team scenarios. A key question is whether these improvements persist under scaling, or whether they degrade as interaction density and joint state-space complexity increase.

    Beyond scalability, it is also important to disentangle the role of A*-derived waypoint information during inference. Although A*-based reward shaping is disabled during evaluation, TA-WAT continues to receive waypoint information as part of its observation space. This introduces a potential confound between training-time reward shaping and inference-time structured observations. Future work should explicitly separate these effects to determine whether performance gains are primarily driven by reward-level supervision, observation-level conditioning, or their interaction, thereby providing a more principled understanding of the mechanisms underlying TA-WAT's robustness.

## 10.5 Conclusion

This thesis investigated the integration of classical A* path planning with deep and multi-agent reinforcement learning for navigation tasks. It addressed four connected questions: (1) whether A*-generated waypoint guidance improves navigation performance across different reinforcement learning algorithms; (2) whether Transformer-based policy architectures can make effective use of structured waypoint information; (3) whether the benefits observed in single-agent A*-guided learning transfer to dual-agent navigation; and (4) whether waypoint-aware Transformer policies remain useful when the task is scaled from single-agent to dual-agent settings.

The results show descriptive differences under A*-guided reward shaping in controlled single-agent settings. In the baseline reproduction and cross-algorithm experiments, A* guidance was associated with higher final-window evaluation-reward values and lower collision-penalty proxy values in several configurations. Evaluation reward is not a direct binary success rate, the collision proxy is not a direct collision rate, and the tests do not establish a general causal improvement in navigation quality. A* guidance also did not consistently improve convergence speed or sample efficiency and introduced additional computational overhead.

The cross-algorithm results further show that the effects of A* guidance are algorithm-dependent. MADDPG, MAPPO, and QMIX responded differently to the same planning-derived reward signal, indicating that A*-guided learning does not transfer uniformly across reinforcement learning paradigms. This suggests that the usefulness of A* guidance depends not only on the availability of waypoint information, but also on the learning mechanism, action space, batch-size configuration, and stability properties of the underlying algorithm.

The architectural experiments indicate that simply replacing an MLP policy with a Standard Transformer Policy Baseline is not sufficient to establish better navigation performance. In the reported comparisons, the Standard Transformer had lower values than the MLP baseline in key descriptive metrics. The Task-Adaptive Waypoint-Aware Transformer showed more promising descriptive trends, including higher final-window evaluation-reward values in some configurations and higher selected metric values relative to the Standard Transformer baseline. However, the five-seed statistical analysis did not establish statistically significant differences after Holm-Bonferroni correction. Therefore, the architectural findings should be interpreted as suggestive rather than conclusive. Waypoint-aware Transformer policies may be a promising direction for structured navigation learning, but stronger empirical validation with more seeds, broader environments, and ablation studies is needed before claiming reliable superiority over MLP-based policies.

The dual-agent experiments reveal an important limitation of directly transferring single-agent A*-guided methods to multi-agent navigation. In the single-agent settings, A* guidance was associated with lower collision-penalty proxy values. This proxy combines agent–obstacle and agent–agent collision penalties and is not a direct collision rate. In dual-agent settings, independently generated A* paths may produce inter-agent conflicts, especially around shared bottlenecks. However, because collision types and explicit path conflicts were not measured separately, this remains a plausible explanation rather than a directly confirmed mechanism. The lower proxy values observed in the single-agent comparisons did not automatically carry over to the dual-agent case.

For multi-agent deployment, independent A* guidance should therefore be treated cautiously, and future work should investigate conflict-aware planning mechanisms such as space-time A*, priority-based planning, joint-cost planning, or coordination-aware reward shaping.

The scaling experiments with Transformer-based policies also show that moving from single-agent to dual-agent navigation changes the role of the architecture. The Task-Adaptive Waypoint-Aware Transformer showed positive aggregate descriptive trends at training iteration 500, especially for MADDPG, while QMIX showed mixed outcomes and MAPPO remained unstable under the tested configuration. These results suggest that waypoint-aware architectures may help some algorithms use structured navigation information in dual-agent settings, but the benefits are not uniform across algorithms and should not be described as universally stable or statistically confirmed. The evidence is best interpreted as an observed algorithm-dependent trend that motivates further testing.

Overall, this thesis shows that the descriptive differences associated with classical planning signals depend on the algorithm, architecture, evaluation setting, and agent count. Some controlled single-agent comparisons are favourable, but the collision-penalty proxy pattern does not automatically transfer to dual-agent navigation. Similarly, waypoint-aware Transformer policies show promising descriptive results, but the current statistical evidence is insufficient to support claims of consistent or significant superiority. The main contribution is therefore a careful characterization of the observed patterns, limitations, and open design questions rather than a universal performance claim.

## Code and Data Availability

The source code, exact Python dependency versions, source-level experiment settings, random-seed definitions, map files, and statistical-analysis scripts supporting this thesis are included in the accompanying project archive and are available from the author upon reasonable request. The archive contains 150 pre-generated maps—50 Easy, 50 Medium, and 50 Hard. All experiments reported in this thesis use only the 50 Easy maps: maps 1–30 for training and maps 31–50 for evaluation. The Medium and Hard maps are retained for future work and are not used in the reported results. The archive also includes the settings for MAPPO, MADDPG, QMIX, the Standard Transformer, and TA-WAT.

# Appendix A

# Baseline MARL Training Procedures

## A.1 Baseline Algorithm Details

This appendix presents the training procedures of the baseline multi-agent reinforcement learning algorithms used in this study. All algorithms follow their original formulations without structural modification and are included for completeness and reproducibility. The main body of the thesis focuses on the proposed A*-guided extensions and architectural explorations.

### A.1.1 QMIX

QMIX is a value-based multi-agent reinforcement learning algorithm that factorizes the joint action-value function into individual agent utilities via a monotonic mixing network, enabling centralized training with decentralized execution [14]. The implementation used in this study follows the standard training procedure proposed in the original work.

**Algorithm 9** Standard QMIX Training Procedure (Rashid et al., 2018)

Initialize agent networks $Q_a(o_a, u_a; \theta)$
Initialize mixing network $Q_{\text{tot}}(\cdot; \theta_m)$
Initialize target networks $\theta^- \leftarrow \theta$, $\theta_m^- \leftarrow \theta_m$
Initialize replay buffer $\mathcal{D}$
**for** each episode **do**
    Reset environment and obtain initial observations $\{o_1, \ldots, o_N\}$
    **for** each timestep $t$ **do**
        Each agent selects action $u_a$ using $\epsilon$-greedy w.r.t. $Q_a(o_a, u_a)$
        Execute joint action $\mathbf{u}$ and observe reward $r$, next state $s'$, and observations $o'_a$
        Store transition $(s, \mathbf{u}, r, s')$ in replay buffer $\mathcal{D}$
    **end for**
**end for**
**for** each training step **do**
    Sample minibatch $(s, \mathbf{u}, r, s')$ from $\mathcal{D}$
    Compute individual Q-values $Q_a(o_a, u_a)$ for each agent
    Compute joint action-value $Q_{\text{tot}}(s, \mathbf{u})$ using the mixing network
    Compute target agent values:

$$Q_a^-(o'_a, \arg\max_{u'_a} Q_a(o'_a, u'_a))$$

    Compute target joint value $Q_{\text{tot}}^-(s', \mathbf{u}')$
    Compute TD target:

$$y = r + \gamma Q_{\text{tot}}^-(s', \mathbf{u}')$$

    Compute loss:

$$\mathcal{L} = (Q_{\text{tot}}(s, \mathbf{u}) - y)^2$$

    Update $\theta$ and $\theta_m$ by minimizing $\mathcal{L}$
    Every $C$ steps update target networks:

$$\theta^- \leftarrow \theta, \qquad \theta_m^- \leftarrow \theta_m$$

**end for**

### A.1.2 Recurrent-MAPPO

Recurrent-MAPPO extends MAPPO by incorporating recurrent neural networks into both the actor and critic to address partial observability in multi-agent environments [16].

**Algorithm 10** Standard Recurrent-MAPPO Training Procedure (Yu et al., 2022)

Initialize policy parameters $\theta$ and critic parameters $\phi$ using orthogonal initialization
Set learning rate $\alpha$
**while** step $\leq \text{step}_{\max}$ **do**
    Initialize data buffer $D = \{\}$
    **for** $i = 1$ to batch_size **do**
        Initialize trajectory $\tau \leftarrow \emptyset$
        Initialize actor RNN hidden states $h_{0,\pi}^{(1)}, \ldots, h_{0,\pi}^{(n)}$
        Initialize critic RNN hidden states $h_{0,V}^{(1)}, \ldots, h_{0,V}^{(n)}$
        **for** $t = 1$ to $T$ **do**
            **for** each agent $a$ **do**
                $p_t^{(a)},\ h_{t,\pi}^{(a)} = \pi(o_t^{(a)}, h_{t-1,\pi}^{(a)}; \theta)$
                Sample action $u_t^{(a)} \sim p_t^{(a)}$
                $v_t^{(a)},\ h_{t,V}^{(a)} = V(s_t^{(a)}, h_{t-1,V}^{(a)}; \phi)$
            **end for**
            Execute joint action $u_t$, observe reward $r_t$, next state $s_{t+1}$, and observations $o_{t+1}$
            Append $(s_t, o_t, h_{t,\pi}, h_{t,V}, u_t, r_t, s_{t+1}, o_{t+1})$ to $\tau$
        **end for**
        Compute advantage estimates $\hat{A}$ on $\tau$ using GAE with PopArt normalization
        Compute reward-to-go $\hat{R}$ and normalize using PopArt
        Split trajectory $\tau$ into chunks of length $L$
        **for** $l = 0, 1, \ldots, T//L$ **do**
            Add $(\tau[l : l + L], \hat{A}[l : l + L], \hat{R}[l : l + L])$ to buffer $D$
        **end for**
    **end for**
    **for** mini-batch $k = 1, \ldots, K$ **do**
        Sample mini-batch $b$ from $D$ containing all agent data
        Restore RNN hidden states for each data chunk
        Update policy parameters $\theta$ using Adam optimizer
        Update critic parameters $\phi$ using Adam optimizer
    **end for**
**end while**

# Bibliography


[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pages 5998–6008, 2017.

[2] Alexander Zai and Brandon Brown. *Deep Reinforcement Learning in Action*. Manning, 2020.

[3] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil A. Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[4] Junior Samuel Lopez-Yepez, Antoine Fagette, Charles Dansereau, and Filipe Carvalhais-Sanches. Leveraging A* pathfinding for efficient deep reinforcement learning in obstacle-dense environments. In *IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 41–46, 2024.

[5] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

[7] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray,

John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744, 2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html`.

[8] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

[9] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39): 1–40, 2016.

[10] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, 2nd edition, 2017.

[11] Matthew Lai, Keegan Go, Zhibin Li, Torsten Kröger, Stefan Schaal, Kelsey Allen, and Jonathan Scholz. RoboBallet: Planning for multirobot reaching with graph neural networks and reinforcement learning. *Science Robotics*, 2025. Early access.

[12] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 38(2):156–172, 2008.

[13] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 2085–2087, 2018.

[14] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 4295–4304, 2018.

[15] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pages 6379–6390, 2017.

[16] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of PPO in cooperative multi-agent games. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, 2022.

[17] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 2974–2982, 2018.

[18] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[19] Sven Koenig and Maxim Likhachev. Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics*, 21(3):354–363, 2005.

[20] Sven Koenig, Maxim Likhachev, and David Furcy. Lifelong planning A*. *Artificial Intelligence*, 155(1-2):93–146, 2004.

[21] Daniil Kirilenko, Anton Andreychuk, Aleksandr Panov, and Konstantin Yakovlev. Transpath: Learning heuristics for grid-based pathfinding via transformers. *arXiv preprint arXiv:2212.11730*, 2022. Accepted to AAAI 2023.

[22] Lucas Lehnert, Sainbayar Sukhbaatar, DiJia Su, Qinqing Zheng, Paul Mcvay, Michael Rabbat, and Yuandong Tian. Beyond A*: Better planning with transformers via search dynamics bootstrapping. *arXiv preprint arXiv:2402.14083*, 2024.

[23] S. Jeevanandham, C. Sivasamy, P. Kokila, M. Sundarrajan, T. Elavarasi, and J. Akshya. Transformer-driven visual intelligence for aerial robotic path optimization. In *2025 5th International Conference on Soft Computing for Security Applications (ICSCSA)*, 2025. doi: 10.1109/ICSCSA66339.2025.11170731.

[24] Kai Guo, Hanjiang Luo, Hang Tao, Rukhsana Ruby, Zhizun Qin, and Kui Liu. Multi-UAVs collaborative search scheme in marine environments using deep reinforcement learning. In *IEEE International Conference on Advanced Cloud and Big Data (CBD)*, pages 39–44, 2023.

[25] Kyriakos G. Vamvoudakis, Yan Wan, Frank L. Lewis, and Derya Cansever. *Handbook of Reinforcement Learning and Control*. Springer, 2021.

[26] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[27] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2010.

[28] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[29] Richard E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[30] Maxim Likhachev and Sven Koenig. Incremental replanning for mapping. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 667–672, 2002.

[31] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[32] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1646–1654, 2016.

[33] Boyu Liu. Comparative analysis of encoder-only, decoder-only, and encoder-decoder language models. In *Proceedings of the 1st International Conference on Data Science and Engineering*, pages 524–530, 2024.

[34] Thomas M. Moerland, Joost Broekens, Aske Plaat, and Catholijn M. Jonker. Model-based reinforcement learning: A survey. *Foundations and Trends in Machine Learning*, 16(1):1–118, 2023.

[35] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 3215–3222, 2018.

[36] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[37] Fadi AlMahamid and Katarina Grolinger. Reinforcement learning algorithms: An overview and classification. In *IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–7, 2021.

[38] Mohit Sewak. Deterministic policy gradient and the DDPG. In *Deep Reinforcement Learning: Frontiers of Artificial Intelligence*, pages 173–184. Springer, 2019.

[39] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 1587–1596, 2018.

[40] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[41] Zhiqing Xiao. DPG: Deterministic policy gradient. In *Reinforcement Learning: Theory and Python Implementation*, pages 289–311. Springer, 2024.

[42] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 1861–1870, 2018.

[43] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 746–752, 1998.

[44] Frans A. Oliehoek and Christopher Amato. *A Concise Introduction to Decentralized POMDPs*. Springer, 2016.

[45] Lixing Jin, Shuai Li, Hung Manh La, Xiao Zhang, and Bo Hu. Proximal policy optimization based dynamic path planning algorithm for mobile robots. *Electronics Letters*, 57(25):968–970, 2021. doi: 10.1049/ell2.12342.

[46] H. Q. Zhang. Optimizing obstacle avoidance path planning for intelligent mobile robots in multi-obstacle environments. *Advances in Production Engineering & Management*, 19(3):358–370, 2024.

[47] Cheng Guo, Xin Wang, Yu Zhang, and Wei Li. An A*-guided deep reinforcement learning framework for efficient and adaptive AGV path planning in digital workshops. *Robotics and Autonomous Systems*, 2025. Early access.

[48] Jun Li, Hanping Xiao, and Zhixiong Liu. A multi-agent deep deterministic policy gradient algorithm integrated with the A* algorithm for multiple-equipment integrated scheduling in automated container terminals. *Engineering Optimization*, 2025. doi: 10.1080/0305215X.2025.2547066.

[49] Wei Zhang, Jian Liu, and Ming Chen. DQN-based path planning with DWA and A* integration for mobile robot navigation. *IEEE Access*, 2025. Early access.

[50] Sakana AI. Continuous thought machines, 2024. Non-peer-reviewed industry research concept.

[51] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.