UNIVERSITÀ
DI TRENTO
Department of
Information Engineering and Computer Science

Doctoral Programme in
Information Engineering and Computer Science

Adaptive Meta-Reasoning for Algorithm Selection in Cognitive Agent Architectures

Aya Kherrour

Advisor
Prof. Paolo Giorgini
Università di Trento

Co-Advisor
Prof. Marco Roveri
Dr. Marco Robol

July 2026

Abstract

Autonomous agents operating in dynamic environments face a fundamental challenge: no single algorithm consistently performs best across all conditions. Whether navigating physical spaces, allocating computational resources, or selecting reasoning strategies, agents that commit to a fixed algorithmic approach at design time may risk underperforming when runtime conditions deviate from initial assumptions. This thesis addresses the problem of enabling autonomous agents to adaptively select the best-performing algorithm at runtime while maintaining the transparency of their deliberative reasoning, using grid-based pathfinding as the experimental domain in which the proposed approach is instantiated and evaluated.

To address this challenge, first, we conduct a comprehensive empirical analysis of six heuristic search algorithms across systematically varied grid-based pathfinding environments, demonstrating that algorithm performance is highly sensitive to environmental characteristics such as grid size, start-goal distance, and obstacle density. Second, we investigate whether this performance variability is learnable, training Random Forest classifiers on environmental features to predict optimal algorithm choices for static problem instances. Models achieved prediction accuracies of up to 84% for memory consumption minimization, confirming that algorithm selection is feasible as a supervised learning problem, while also exposing a fundamental limitation: static classifiers cannot revise their choices once execution begins and environmental conditions evolve.

To overcome this limitation, we propose a novel agent architecture that embeds adaptive algorithm selection within the agent's deliberation cycle rather than treating it as an external preprocessing step. The architecture extends the Belief-Desire-Intention (BDI) framework with a reinforcement learning meta-reasoner, implemented as a Deep Q-Network, that continuously monitors the agent's belief–intention state and dynamically selects among planning algorithms at runtime. Algorithm selection is formalized as a sequential decision-making problem extending Rice's classical framework to dynamic settings, where states encode environmental observations and deliberative context, actions correspond to algorithm choices, and rewards measure the quality and efficiency of planning outcomes.

Experimental evaluation in dynamic pathfinding environments with moving obstacles demonstrates that the trained meta-reasoner achieves a success rate of 99.43% and

substantially lower penalties compared to all fixed-algorithm baselines, with the weakest baseline reaching only 88.83% success at over ten times the penalty cost. The agent learns a behaviourally interpretable and context-sensitive strategy, favouring deeper lookahead in sparse environments and switching to faster replanning as obstacle density increases. These results confirm that integrating learned meta-level reasoning into a cognitive architecture yields tangible performance advantages over static approaches while preserving the transparency and goal-directed structure of the BDI deliberation layer.

Keywords

BDI Agents, Reinforcement Learning, Algorithm Selection, Meta-Reasoning, Adaptive Decision-Making, Cognitive Architectures

Acknowledgments

This thesis is the result of years of work, carried out with perseverance and, above all, trust in God First and foremost, I am grateful to my supervisors, Prof. Paolo Giorgini, Prof. Marco Roveri, and Dr. Marco Robol, for their intellectual guidance, critical insight, and continued engagement throughout this research. Their feedback sharpened my thinking at every stage, and their trust in my work gave me the space to develop it on my own terms.

I would also like to thank the University of Trento and the Department of Information Engineering and Computer Science for the institutional and financial support that made this research possible, and for fostering an environment in which independent inquiry could thrive. To my dear family—my mother, my father, and my siblings—thank you for the unconditional love and unwavering support that made this journey possible. A special thank you to those who took the time to call, to check in, and to encourage me when it mattered most. Your presence, even across the distance, was a constant source of strength.

To every friend, colleague, flatmate, and well-wisher I have encountered along the way, thank you for every conversation over coffee, the small moments of connection, the ordinary days that turned out to matter more than I expected. Thank you for being part of this.

Per l'italia e agli italiani grazie per l'accoglienza vi voglio bene, pour l'Algérie tu es toujours dans mon cœur.

Above all, praise and gratitude belong to Almighty God for the strength, the clarity, and the resilience that carried me through.

Contents

List of Tables

List of Figures

List of Abbreviations

AI	Artificial Intelligence
ARA*	Anytime Repairing A*
BDI	Belief-Desire-Intention
CNN	Convolutional Neural Network
D*	D Star
DQN	Deep Q-Network
LLM	Large Language Model
LPA*	Lifelong Planning A*
LRTA*	Learning Real-Time A*
MAPF	Multi-Agent Path Finding
MDP	Markov Decision Process
ML	Machine Learning
RF	Random Forest
RL	Reinforcement Learning
RTAA*	Real-Time Adaptive A*
SAT	Satisfiability
TSP	Traveling Salesman Problem

Chapter 1

Introduction

Autonomous agents are increasingly being deployed in real-world settings where conditions evolve unpredictably. This includes physical systems such as robots navigating dynamic workspaces, drones adjusting their routes to sudden obstacles, and autonomous vehicles reacting to changing traffic patterns.

This challenge is not confined to physical systems. It extends to emerging classes of LLM-based agents [57] that operate in complex digital environments, and are now being used to analyse documents, coordinate workflows, or assist in decision-making tasks where the surrounding context can change from one moment to the next. Whether the agent is a robot confronted with moving obstacles or an LLM agent selecting the most appropriate tool or retrieval strategy, the underlying challenge is similar: no single algorithm or decision-making mechanism is effective across all situations.

In many domains, multiple algorithms can solve the same underlying problem, yet their performance can vary substantially depending on factors such as environment characteristics, user demand, and available computational resources [23, 24, 4, 25]. As a result, an agent that commits to a single algorithm at design time and cannot revise this choice during execution will inevitably underperform when runtime conditions deviate from initial assumptions. In pathfinding, for example, the performance of different planning algorithms varies greatly depending on the characteristics of the environment. Classical A* [18] is well suited for static settings, while its extensions, such as D* [51], D* Lite [26], and LPA* [28], are more appropriate when the environment changes during execution. Analogously, LLM agents often need to choose between multiple strategies, such as different retrieval methods, reasoning modes, or planning routines, depending on the nature and complexity of the task at hand. Load balancing algorithms, such as round-robin, least connections, and weighted balancing, are widely used in

cloud environments. However, cloud environments are highly dynamic due to factors such as user demand, network traffic, or application behavior. This variability often leads to suboptimal performance such as server overload, increased response times, and inefficient resource utilization. In all of these settings, the fundamental limitation is the same: static algorithm selection cannot account for the variability that emerges once an agent begins operating in a changing environment.

Previous investigations have shown that algorithm performance is highly sensitive to environmental conditions, leaving no single approach consistently superior [23, 24, 4, 7, 29]. While these works focus on analysing performance variability across static problem instances, they do not address how agents can autonomously revise algorithmic choices during execution. Recent work such as [46, 43, 2] has explored learned algorithm selection in multi-agent pathfinding, but without integrating such capabilities into a deliberative agent architecture. Critically, existing approaches to learning-enhanced Belief–Desire–Intention (BDI) architectures, such as [48, 49, 56, 15], focus on plan selection or goal-level policy learning; they do not treat algorithm choice itself as a dynamic, sequential decision-making problem embedded within the agent's deliberation cycle. This constitutes a clear gap: no existing framework enables a BDI agent to adaptively select among alternative algorithms at runtime, based on evolving environmental observations, while preserving the transparency and goal-directed reasoning that deliberative architectures are designed to provide [46, 43, 2, 48, 49]. Addressing this gap is the central motivation of the present thesis.

To address this limitation, this thesis proposes a principled framework for adaptive algorithm selection within deliberative agent architectures. Pathfinding serves as the experimental domain throughout, providing a well-defined, reproducible setting in which to instantiate and evaluate the proposed approach. Although the framework is formulated in general terms and is not tied to assumptions specific to this domain, its empirical validation is limited to pathfinding; extending it to other domains is left as future work (see Chapter 7).

1.1 Problem

The central problem addressed in this thesis is how to enable autonomous agents, operating in dynamic settings, to adaptively select the best-performing algorithm at runtime, keeping the reasoning context in which those decisions are made transparent and inspectable. This challenge encompasses several interconnected issues. First, agents must be aware of environmental changes that may necessitate switching from

one algorithm to another. Second, they must know which algorithms perform best under different conditions without relying on exhaustive, manually defined selection rules. Third, the agent's architecture must support this adaptive behavior while preserving the transparency of its deliberative reasoning, so that human operators can inspect and trust the context in which algorithmic decisions are made.

The Belief–Desire–Intention (BDI) model [42] provides a transparent and goal-oriented structure for autonomous reasoning, yet its hard coded nature limits its learning and adaptability in uncertain environments. In contrast, Reinforcement Learning (RL) [54], enables agents to learn context-specific behaviours through a series of trial and error, however, at the cost of losing interpretability due to its black-box nature. Consequently, the challenge is to combine the strengths of both paradigms in such a way as to preserve the structured reasoning and transparency of BDI while incorporating the adaptive learning capabilities of RL.

1.2 Research Questions

Research questions:

1. To what extent does algorithm performance depend on environmental characteristics, and which characteristics drive this variability?

   Algorithm performance is known to vary substantially across problem instances, yet this variability is often treated as incidental rather than systematically analysed. In domains such as pathfinding, environmental characteristics including grid size, obstacle density, and start–goal distance can significantly affect the efficiency, memory consumption, and success rate of different planning algorithms. Understanding how performance depends on such characteristics is a necessary first step toward informed algorithm selection. This research question investigates the degree of performance variability across a range of environments and identifies the key environmental factors that most strongly influence algorithm behaviour. We address this question in Chapter 3.

2. Can learning-based approaches use environmental features and experience to predict and adapt optimal algorithm selection strategies?

   Given that algorithm performance depends on observable characteristics of the environment, learning-based methods may be used to predict which algorithm is most suitable for a given problem instance. Supervised learning approaches can

exploit environmental features to perform algorithm selection when conditions remain fixed. However, in dynamic environments, execution conditions may change after planning begins, limiting the effectiveness of static predictions. This research question examines the extent to which learning-based approaches can exploit environmental features and execution experience to support algorithm selection, and analyses their limitations in settings where adaptation during execution is required. Addressed in Chapter 4.

3. How can adaptive algorithm selection be realized in autonomous agent systems while preserving transparency of the agent's architecture and interpretability of the conditions under which selections are made?

   While learning-based algorithm selection has been explored in isolation, integrating such capabilities into autonomous agent systems raises additional challenges. In deliberative agents, algorithm choice affects not only performance but also the transparency of the agent's reasoning process and the interpretability of the conditions that inform its decisions. This research question focuses on how adaptive algorithm selection can be embedded within an autonomous agent architecture, specifically a Belief–Desire–Intention (BDI) framework, such that algorithmic decisions are made at runtime while maintaining the transparency, structure, and goal-directed reasoning characteristic of deliberative agents. Addressed in Chapters 5 and 6.

4. Does adaptive algorithm selection improve the agent's performance compared to static selection strategies in dynamic environments?

   Ultimately, adaptive algorithm selection is only justified if it leads to measurable benefits over static approaches. This research question evaluates whether adaptive, runtime algorithm selection leads to improved performance compared to fixed algorithm strategies, considering metrics such as success rate in dynamic pathfinding environments. Addressed in Chapter 6.

1.3 Research contributions

The contributions of this research include the following.

1. **Empirical Characterization of Algorithm Performance Variability in Pathfinding** The thesis provides a comprehensive analysis of performance variability of multiple heuristic search algorithms solving variant pathfinding scenar-

ios. The results demonstrate that there is no single dominant algorithm for all scenarios. These findings establish both the potential and fundamental limitations of static algorithm selection, motivating the need for adaptive and learning-based, runtime approaches [23, 24]

2. **A Formal Framework for Sequential Algorithm Selection in Dynamic Environments** We extend Rice's algorithm selection framework [44] from static, single-shot decisions to dynamic, sequential settings by formulating algorithm choice as a Markov Decision Process. This formalization enables agents to continuously revise algorithmic decisions as conditions evolve, addressing a fundamental gap in classical algorithm selection theory that assumes static problem instances.

3. **An Adaptive BDI Agent Architecture with Learned Meta-Reasoning** This thesis proposes an extended Belief–Desire–Intention (BDI) agent architecture augmented with a reinforcement learning-based meta-reasoning component that enables runtime algorithm selection. The meta-reasoner is invoked by the means–end reasoning process, selecting among alternative planning algorithms based on the agent's current beliefs and intentions while preserving the deliberative transparency and goal-directed structure of the BDI layer. Empirical evaluation in dynamic pathfinding environments with moving obstacles shows that the proposed architecture outperforms static algorithm selection strategies in the evaluated scenarios, achieving improved robustness and execution efficiency [22].

1.4 Structure of the Thesis

The rest of the manuscript is structured as follows:

- **Chapter 2:** This chapter presents an overview of the foundational concepts including cognitive agent architecture, algorithm selection, and establishes our contributions relative to the existing literature.

- **Chapter 3:** Provides a comprehensive empirical analysis of six pathfinding algorithms in various 2D grid-based scenarios. Addressing **RQ1**, this chapter investigates the impact of different environmental features on multiple algorithm performance metrics. The results demonstrate that algorithm performance varies significantly across scenarios, with no algorithm consistently superior.

- **Chapter 4:** Describes a supervised learning approach for algorithm selection in pathfinding. This chapter presents the data collection process, feature engineering combining grid-based features, such as obstacle density and map size, and CNN-based features from map images, and the training of predictive models. Each model optimises a specific performance metric such as memory consumption, computational speed, and path quality. Addressing **RQ2**, the results show that environmental features can effectively predict optimal algorithm choices for specific scenarios.

- **Chapter 5:** Presents the thesis's core contribution: a cognitive agent architecture that integrates meta-reasoning for runtime algorithm selection. The chapter addresses **RQ3** by identifying the architectural requirements, justifying the choice of BDI framework and reinforcement learning as the learning mechanism, and describing their integration.

- **Chapter 6:** Translates the proposed architecture from concept to practical execution. This chapter presents the implementation and training of the meta-reasoner within the BDI agent, using a pathfinding case study. The learned policy maps environment states, derived from current beliefs and intentions, to algorithm choices. Addressing **RQ3** and **RQ4**, the experimental results demonstrate that adaptive selection outperforms static selection strategies in dynamic environments.

- **Chapter 7:** Summarizes the key contributions of the thesis and synthesizes how the proposed solution enhances the performance of autonomous agents in dynamic environments. In addition, it discusses the limitations of this work and outlines possible directions for future research.

1.5 Publications

The contributions of this thesis are grounded in the following peer-reviewed publications. The first publication constitutes the initial empirical study, which was subsequently extended and published as the second (journal) contribution; together they form the foundation of Chapters 3 and 4. The third publication presents the proposed BDI architecture augmented with a reinforcement learning meta-reasoner and its instantiation in a dynamic pathfinding domain, corresponding to Chapters 5 and 6.

- Aya Kherrour, Marco Robol, Marco Roveri, and Paolo Giorgini. *Evaluating Heuristic Search Algorithms in Pathfinding: A Comprehensive Study on Perfor-*

mance Metrics and Domain Parameters. In Proceedings of the Third Workshop on Agents and Robots for reliable Engineered Autonomy (AREA@ECAI 2023), EPTCS, pages 102–112, 2023.

- Aya Kherrour, Marco Robol, Marco Roveri, and Paolo Giorgini. *A Multi-Algorithm Pathfinding Method: Exploiting Performance Variations for Enhanced Efficiency*. Annals of Mathematics and Artificial Intelligence, 93(4):569–588, 2025.

- Aya Kherrour, Rawlings Ntassah, Marco Robol, Marco Roveri, and Paolo Giorgini. *RL-Driven Meta-Reasoning for BDI Agents*. In Artificial Intelligence and Soft Computing – ICAISC 2026, Lecture Notes in Artificial Intelligence, Springer, 2026. *(To appear.)*

Chapter 2

Background and Related Work

This chapter surveys the foundational concepts and prior research that underpin the contributions of this thesis. The work sits at the intersection of four closely related research areas: *(i)* the classical algorithm selection problem and its meta-reasoning extensions; *(ii)* the empirical study of algorithm performance variability in pathfinding domains; *(iii)* the theory and practice of cognitive agent architectures, in particular the Belief-Desire-Intention (BDI) model augmented with machine learning; and *(iv)* reinforcement learning and Deep Q-Networks, which provide the adaptive learning mechanism that ties the architectural contributions together. Two dimensions recur across these areas and provide the analytical thread of the review: the *temporal regime* of algorithm selection — whether the choice is made once before execution (static) or revised continuously at runtime (dynamic), and the *selection level* at which it operates, whether over symbolic plan templates (plan-level) or over the algorithmic implementations of means-end reasoning themselves (algorithm-level). Each area is reviewed in turn along these two dimensions, and the chapter concludes by drawing them together in Table 2.1 to identify the precise gap in the existing literature that this thesis addresses.

2.1 The Algorithm Selection Problem and Meta-Reasoning

In many domains, a collection of algorithms may be available for solving the same underlying task, yet no single algorithm consistently outperforms the others across all problem instances. This observation was first formalised by Rice [44], whose seminal framework defines the algorithm selection problem as the task of mapping each problem instance — characterised by a set of observable features — to the algorithm expected to yield the best performance according to some criterion. Rice's core insight is that

rather than searching for a universally superior algorithm, one should instead learn a selection mapping that exploits the relationship between instance features and algorithm behaviour. This formulation has since become the theoretical foundation for a broad family of approaches in automated algorithm configuration and selection, surveyed comprehensively by Kerschke et al. [21]. The algorithm selection problem has been studied extensively in combinatorial optimisation, satisfiability solving, and machine-learning hyperparameter tuning. One influential realisation of Rice's framework is the *portfolio-based* approach, in which a scheduler maintains a set of heterogeneous solvers and allocates computation among them: the SAT solver SATzilla [59] is a canonical example, and the same principle underlies model-based configuration methods such as sequential model-based algorithm configuration (SMAC) [19]. These methods demonstrate that substantial performance gains can be obtained simply by selecting the most appropriate algorithm for each instance. Their effectiveness, however, rests on feature engineering: the discriminative power of a selection model depends critically on whether the extracted features capture the properties that actually drive performance variability [50]. A closely related tradition is *meta-reasoning*, which treats the selection and scheduling of reasoning strategies as a first-class decision problem: a rational agent should decide not only which action to take in the world, but also which cognitive operation to perform next, weighing the expected value of further deliberation against its computational cost [45]. Portfolio selection and meta-reasoning thus share a common premise — that the choice of *how* to compute is itself a decision amenable to principled optimisation — and together they form the conceptual foundation on which this thesis builds.

Despite the maturity of these lines of work, a fundamental limitation of classical algorithm selection theory is its assumption that the problem instance remains static throughout execution. Rice's original framework [44] considers only single-shot, per-instance selection: the algorithm is chosen once before execution begins and is not revised thereafter. This assumption is adequate for static combinatorial problems but breaks down in dynamic settings — such as pathfinding in environments with moving obstacles — where the characteristics of the problem evolve during execution and the optimal algorithm choice may change accordingly. The extension of algorithm selection to sequential, dynamic settings where choices must be revised continuously as new information becomes available remains an open problem and constitutes one of the central theoretical contributions of this thesis.

2.2 Algorithm Performance Variability in Pathfinding

Pathfinding in grid-based environments is a well-studied problem with broad practical relevance in robotics, autonomous navigation, and video games [1, 36]. The task consists of computing a path from the initial to a goal location while avoiding obstacles. A wide variety of algorithms have been proposed, including classical graph search methods such as A* [18] and Dijkstra's algorithm [13], incremental and replanning algorithms such as D* [51], D* Lite [26], and LPA* [28], real-time heuristic search algorithms such as LRTA* [30] and RTAA* [27], and anytime algorithms such as ARA* [33]. Each of these algorithms has different trade-offs between solution quality, computational cost, memory consumption, and responsiveness to environmental change, making the selection of an appropriate algorithm for a given setting a non-trivial task.

Several studies have examined how pathfinding algorithms behave across different grid environments, though they vary in focus and methodology. Lawande et al. [32] conducted a systematic review and analysis of intelligence-based pathfinding algorithms in video game settings, demonstrating that grid size and obstacle placement significantly influence execution time and the number of expanded nodes. Zarembo and Kodors [60] approach the problem from a different angle, by studying the efficiency of pathfinding algorithms on two-dimensional grids of varying sizes and confirming that grid size is a primary determinant of computational cost. Permana et al. [39] performed a comparative analysis of A*, Dijkstra's algorithm, and breadth-first search in a maze runner game, finding that algorithm performance varies systematically across game levels that differ in obstacle configuration. Similar observations appear in the work of Iloh [20] who evaluates DFS, BFS, and A* in maze traversal tasks and finds that no single method consistently performs best across all configurations. In parallel, Sturtevant [53] introduced standardised benchmarks for grid-based pathfinding, providing a principled basis for comparing algorithm performance across diverse map types and facilitating reproducible evaluation.

As part of this doctoral research, and developed in full in Chapter 3, we conducted a comprehensive evaluation of six heuristic search algorithms [23, 24], namely D*, D Lite, LPA, LRTA*, RTAA*, and ARA*, across a systematically varied set of grid environments differing in size, obstacle density, and start-goal distance. The obtained results revealed that grid size turns out to be the dominant factor affecting both memory usage and solving time. D* Lite, in particular, tends to remain more memory-efficient across most settings. Start-goal distance showed a strong positive correlation with path length and a moderate correlation with solving time, with ARA* performing best for

short distances and RTAA* excelling at longer distances. Obstacle density, contrary to initial expectations, exhibited a comparatively modest influence on most performance metrics, though D*, ARA*, and LPA* showed increased path lengths in dense environments. Overall, no single algorithm consistently achieves the best performance across all conditions and all metrics, establishing the empirical foundation for the algorithm selection approach pursued in this thesis.

Building on this evaluation, and presented in full in Chapter 4, a second stage of this doctoral research extended the analysis to the Multi-Agent Pathfinding (MAPF) setting [24]: we incorporated benchmark maps from the MAPF literature [52] and trained machine learning models to predict the best-performing algorithm for a given instance based on both handcrafted grid features, such as obstacle density, map size, and start-goal distance, and features extracted from map images using a Convolutional Neural Network [47]. Random Forest classifiers [10] trained on this feature set achieved prediction accuracies of up to 84% for the memory consumption criterion, confirming that algorithm selection is feasible as a supervised learning problem. The results further show that combining handcrafted features with those extracted by a CNN tends to improve predictive performance compared to using either representation on its own. This suggests that richer, multi-modal descriptions of the environment are useful for algorithm selection.

Work in the broader MAPF literature follows a similar direction, though with different design choices. For instance, Ren et al. [43] introduce MAPFAST, where environment structure, obstacle distribution, agent positions, and shortest-path information are encoded jointly to predict a suitable MAPF solver. In contrast, Alkazzi et al. [2] focus on efficiency, introducing MAPFASTER, a faster and simpler approach to MAPF algorithm selection that reduces the overhead of the selection process while maintaining competitive prediction accuracy. Other approaches move beyond direct selection. Damani et al. [12], for example, explore deep reinforcement learning and imitation learning for multi-agent lifelong pathfinding, showing that learned policies can effectively guide navigation and replanning in multi-agent settings. Parooei et al. [37] instead explored a decentralised approach to MAPF using scalable feature fusion, combining features across multiple spatial representations (1D, 2D, and 3D) to enable scalable, collision-free planning. Overall, these works highlight a common trend: the algorithm's performance depends strongly on the specific instance, and learning-based methods are well-suited to capturing this variation and guiding the selection process.

Both our own supervised selector in Chapter 4 and the supervised MAPF selectors discussed above occupy the same position in the design space: they perform algorithm-

level selection but treat it as a *static*, per-instance decision fixed before execution begins. This is precisely the static, algorithm-level region later summarised in Table 2.1. What none of them provides is the ability to *revise* that choice as the environment changes, a limitation that becomes decisive in dynamic settings, where obstacles move and previously computed paths may be invalidated, so that an algorithm well suited to the initial conditions may cease to be the best choice as execution proceeds. Overcoming this limitation, by reformulating algorithm selection as a sequential decision embedded in the agent's reasoning cycle, is the departure that distinguishes the novel contribution of this thesis from the earlier stages that motivate it. Realising that reformulation concretely requires a well-defined set of alternatives to select among; the family of heuristic search algorithms considered throughout this thesis is introduced next.

2.3 Heuristic Search Algorithms for Dynamic Environments

The pathfinding algorithms considered in this thesis represent a range of approaches for navigating grid-based environments. They differ in several aspects, including solution quality, computational effort, memory usage, and responsiveness to changes in the environment. In the following, we provide an overview of the key ideas needed to interpret the experimental results presented later.

A* [18], proposed by Hart, Nilsson, and Raphael, is the classical benchmark for heuristic graph search. It computes optimal paths by combining a cost-so-far estimate with an admissible heuristic, guaranteeing both completeness and optimality in static environments. Its limitations in dynamic and resource-constrained settings motivated the development of other algorithms.

D* (Dynamic A*) [51] was among the first algorithms designed to handle dynamic environments. Instead of recomputing paths from scratch, D* updates its internal cost representation incrementally as obstacles appear or disappear. This allows for efficient replanning, although the need to maintain back-pointers for many nodes leads to relatively high memory consumption, especially in large grids.

LPA* (Lifelong Planning A*) [28] an incremental extension of A*. Rather than discarding previous computations, it reuses information and propagates only the updates required when edge costs change. This makes it effective in environments where modifications are frequent but local. One limitation, however, is that it assumes a fixed start position, which restricts its applicability in scenarios where the agent's starting point

varies.

D* Lite [26] simplifies the original D* approach while preserving its core behaviour. It relies on ideas closely related to LPA*, resulting in a cleaner formulation that is generally easier to implement. In practice, it achieves comparable solution quality with lower overhead, making it well-suited for goal-directed navigation in changing or partially unknown environments.

LRTA* (Learning Real-Time A*) [30] follows a different approach by interleaving planning and execution. Instead of computing a full path in advance, it selects actions greedily based on the current heuristic and updates that heuristic online as new information becomes available. Over time, this learning process improves decision quality, although early solutions are often suboptimal.

RTAA* (Real-Time Adaptive A*) [27] adds a bounded lookahead search at each decision step. The depth of the lookahead plays an important role, in which shallow searches (i.e., lower lookaheads) are faster but less informed, whereas deeper searches (i.e., larger lookaheads) produce better decisions at a higher computational cost. Managing this trade-off is central to the meta-reasoning approach explored in this thesis.

ARA* (Anytime Repairing A*) [33], proposed by Likhachev, Gordon, and Thrun, adopts an anytime strategy. First, it computes a solution quickly using an inflated heuristic, and then gradually refines it as more computation becomes available by gradually reducing the inflation factor. This makes it particularly useful when a quick initial solution is required, but solution quality can be improved incrementally.

These algorithms are best understood not as rivals for a single title but as points on a shared trade-off surface: optimality and completeness in static settings (A*), incremental replanning under change (D*, D Lite, LPA), bounded-time responsiveness (LRTA*, RTAA*), and anytime refinement (ARA*). Which point is preferable is not fixed in advance; it depends on the size of the environment, the rate at which it changes, and the time and memory available at the moment of decision. This is precisely what turns algorithm choice into a genuine decision rather than a design-time default — and it raises the question of *where*, within an autonomous agent, such a decision should be made and revised. The following section introduces the cognitive architecture in which we situate that decision.

2.4 Cognitive Agent Architectures and the BDI Model

Among the cognitive agent architectures proposed in the multi-agent systems literature, the Belief-Desire-Intention (BDI) model [42] is one of the most widely adopted, offering

a structured and interpretable framework for autonomous reasoning.

The BDI model, formalised by Rao and Georgeff [42], represents an agent's mental state in terms of three interconnected components. *Beliefs* encode the agent's current understanding of its environment, updated incrementally as new percepts arrive through a belief revision function. *Desires* represent the set of potential objectives the agent might pursue, while *intentions* correspond to the goals to which the agent has committed and for which it is actively constructing or executing a plan. Decision-making in a BDI agent unfolds through two complementary processes: *deliberation*, in which the agent selects which desires to adopt as intentions based on its current beliefs and prior commitments, and *means-end reasoning*, in which the agent identifies or generates a plan that, if executed, would bring about the selected intention. Traditional BDI systems implement means-end reasoning through a static plan library: a repository of pre-authored plan templates, each associated with a triggering condition and a context condition that determines its applicability at runtime. The Jason programming language [8] represents a widely used and well-regarded implementation of this paradigm. When an intention is adopted, the agent searches its plan library for a plan whose context condition is satisfied by the current belief state, selects among applicable plans, and commits to executing the chosen plan. This design offers several important advantages: it supports runtime responsiveness through deferred plan selection, it provides built-in mechanisms for handling plan failure, and it preserves a transparent, symbolic representation of the agent's reasoning process that is comprehensible to human designers and operators. However, the reliance on static plan libraries and manually authored context conditions imposes a significant limitation: the agent's adaptability is bounded by the completeness and correctness of its plan library, which is typically designed at system build time under assumptions about the environment that may not hold at runtime. In environments characterised by unpredictable or non-stationary dynamics, including pathfinding environments with moving obstacles, a BDI agent with a fixed plan library may repeatedly select suboptimal plans, or fail to find applicable plans altogether, when runtime conditions deviate from designer assumptions. Recognising this limitation, a substantial body of research has investigated mechanisms for enhancing BDI adaptability through the integration of machine learning techniques [49, 56, 5, 40, 15]. Efforts along these lines fall into two broad traditions: learning better selection among existing plans, and integrating reinforcement learning directly into the reasoning cycle itself. Both are reviewed in turn below, before identifying what neither provides.

2.4.1 Plan Selection Approaches

A first and influential line of work focuses on improving how BDI agents select among available plans in their library. Rather than relying solely on manually specified context conditions, these approaches augment the selection process with learned preferences derived from execution experience. Singh et al. [48] extended the BDI model with a probabilistic plan selection mechanism that uses execution history to weight alternative plans according to their empirical success rates in specific contexts. Their approach maintains a probability distribution over applicable plans for each goal-context pair and updates these distributions through online learning after each plan execution. The resulting selection strategy naturally balances exploration of less-tested plans with exploitation of plans known to succeed, and supports continuous adaptation as execution conditions change. The same group later [49] demonstrated that learned plan preferences can be updated online to track shifts in the environment without requiring manual re-specification of context conditions, extending the approach to environments with changing dynamics. Faccin and Nunes [16] proposed a complementary approach based on predictive meta-modelling. Rather than learning from post-execution outcomes, their method trains a model to predict plan outcomes *before* execution begins, allowing the agent to proactively select the plan most likely to succeed given the current context. This anticipatory selection strategy reduces the cost of plan failures by avoiding poor plans before they are attempted, at the expense of requiring a predictive model that must be trained offline or updated incrementally. Erduran [15] explored the use of multiple machine learning algorithms, including decision trees, naive Bayes classifiers, and neural networks, for cognitive and autonomous BDI agents, examining the trade-offs between predictive accuracy and runtime overhead across different learning paradigms. These approaches collectively demonstrate that learning plan preferences from experience can substantially enhance BDI adaptability. However, they share a common architectural assumption: they take the plan library as given and focus on selecting *which* pre-authored plan to execute. They do not address the question of *how* a BDI agent should select among algorithmic implementations — that is, which computational procedure should be invoked to realise a given plan, a distinction that is central to the meta-reasoning contribution of this thesis.

2.4.2 Reinforcement Learning Integration in BDI Architectures

A second, more recent line of research integrates reinforcement learning directly into the BDI reasoning cycle, enabling agents to learn behaviours that go beyond the scope of

any pre-authored plan library. Bosello and Ricci [9] introduced Jason-RL, a framework that augments the Jason BDI reasoning cycle with an RL-based component that allows agents to learn so-called *soft plans*: behaviourally grounded action sequences discovered through interaction with the environment rather than specified by a designer. Jason-RL enables a principled integration of symbolic and subsymbolic reasoning within a single agent, with learned soft plans complementing the existing library of symbolic plans. However, the approach is evaluated in single-agent scenarios, which may limit its generalisation to multi-agent or highly dynamic settings. Pulawski, Dam, and Ghose [40] proposed BDI-Dojo, a framework that uses adversarial reinforcement learning to train BDI agents capable of maintaining robust behaviour under dynamic and perturbed conditions. By training agents against adversarially chosen perturbations, BDI-Dojo produces policies that are more resilient to unexpected changes in the environment. While this approach demonstrates how RL can strengthen the behavioural robustness of BDI agents, its focus remains on learning robust policies for fixed plan execution rather than supporting meta-level decisions about which algorithm to invoke. Wan et al. [56] proposed an extension of the BDI model that incorporates Q-learning to support decision-making in uncertain environments. Their approach augments the BDI deliberation cycle with a Q-table that estimates the long-term value of committing to different intentions given the current belief state, allowing the agent to adapt its intention selection strategy through experience. Badica et al. [5] similarly explored the application of temporal difference learning within agent-oriented programming frameworks, demonstrating that value-based RL methods can be integrated into the BDI cycle without disrupting its symbolic structure. Taken together, these contributions establish that reinforcement learning can be productively integrated into BDI architectures to overcome the rigidity of static plan libraries. However, a critical gap remains: none of these approaches addresses the problem of algorithm selection at the level of *means-end reasoning*. Existing RL-BDI integrations focus on learning policies for selecting among symbolic plans or intentions, not on selecting among algorithmic implementations of the planning process itself. This distinction implies a different state representation, a different action space, and a different formulation of the learning problem, all of which are developed in this thesis.

2.5 Reinforcement Learning and Deep Q-Networks

Reinforcement learning (RL) provides a general framework for learning sequential decision-making from interaction with an environment [54]. An agent repeatedly observes the

current state, selects an action according to its policy, and receives a scalar reward that reflects the outcome of that decision. This feedback is then used to adjust the policy over time. The overall objective is to maximise the expected cumulative discounted reward, which requires balancing short-term gains against longer-term effects.

The Markov Decision Process (MDP) formalises this interaction [54]. It is defined by a state space $\mathcal{S}$ and an action space $\mathcal{A}$, together with a transition function $T(s, a, s')$ describing the probability of moving from s to s' after taking action a, and a reward function $R(s, a, s')$ that assigns an immediate value to each transition. A discount factor $\gamma \in [0, 1)$ determines how future rewards are weighted relative to immediate ones. Within this setting, the action-value function $Q^\pi(s, a)$ captures the expected return of taking action a in state s and then following policy π. Q-learning aims to estimate the optimal function $Q^*(s, a)$ directly from experience, without relying on a model of the environment dynamics.

Deep Q-Networks (DQNs), introduced by Mnih et al. [34], extend this idea to high-dimensional state spaces by representing the action-value function with a neural network parameterised by weights θ. Instead of storing values in a table, the network approximates $Q(s, a; \theta)$, which allows it to generalise across similar states and scale to more complex environments. Two design choices are particularly important. First, *experience replay* stores past transitions and samples them in mini-batches, reducing correlations in the training data. Second, a *target network*—a periodically updated copy of the main network—helps stabilise the learning targets. These modifications were key to the success of DQNs, which achieved strong performance on Atari games directly from pixel inputs.

A standard way to balance exploration and exploitation in these methods is the ε-greedy strategy. With probability ε, the agent selects a random action, while with probability $1-\varepsilon$ it chooses the action with the highest estimated Q-value. In practice, ε is gradually reduced during training: it starts relatively high to encourage exploration and decreases as the policy becomes more reliable.

2.6 Summary and Research Gap

The literature reviewed in this chapter establishes a rich set of foundations across four interconnected research areas, yet also reveals a clear and consistent gap that this thesis is designed to address. From algorithm selection theory [44], the field inherits the core insight that performance-aware algorithm choice can substantially outperform fixed, universal algorithmic strategies. From empirical studies of pathfinding algorithm per-

formance [24, 23, 32, 39, 60, 53], it is well established that no single algorithm dominates across all environmental conditions, and that observable features such as grid size, obstacle density, and start-goal distance are informative predictors of relative algorithm performance. From the machine learning for algorithm selection literature [43, 2, 24], supervised and deep learning approaches have demonstrated the feasibility of automating these selection decisions based on environment representations, achieving competitive prediction accuracy in both single-agent and multi-agent settings. From the BDI literature [42, 48, 49, 16, 9, 40, 56], a well-developed set of approaches establishes that learning can be integrated into the BDI reasoning cycle to improve plan selection and behavioural robustness, while preserving the transparency and goal-directed structure that are among the most valued properties of deliberative agent architectures. From the reinforcement learning literature [54, 34], the Deep Q-Network framework provides a scalable and well-understood mechanism for learning sequential decision-making policies from reward signals, with well-established techniques for stabilising training in complex, high-dimensional state spaces. Despite this breadth of existing work, a critical gap persists at the intersection of these areas: no existing approach addresses adaptive algorithm selection as a meta-reasoning process embedded *within* the BDI deliberation cycle. The precise nature of this gap, and how the surveyed literature falls short of it, is detailed below.

Along the two dimensions introduced at the outset of this chapter temporal regime and selection level, the surveyed literature falls into two disjoint groups, positioned explicitly in Table 2.1 below.

Table 2.1: Comparative summary of related work along the two dimensions central to this thesis. *Temporal regime* indicates whether the selection decision is made once before execution (static) or revised continuously at runtime (dynamic). *Selection level* refers to whether the approach operates over symbolic plan templates (plan-level) or over algorithmic implementations of means-end reasoning (algorithm-level). *Learning mechanism* characterises the method used to inform selection, if any.

Work	**Domain**	**Temporal Regime**	**Selection Level**	**Learning Mechanism**
Rice [44]	General	Static	Algorithm-level	None (framework)
Kherrour et al. [24] (this thesis, Chs. 3 4)	Pathfinding/MAPF	Static	Algorithm-level	Supervised (RF, CNN)
Ren et al. [43]	MAPF	Static	Algorithm-level	Supervised (MAPFAST)
Alkazzi et al. [2]	MAPF	Static	Algorithm-level	Supervised (MAPFASTER)
Damani et al. [12]	MAPF	Dynamic	Algorithm-level	RL / Imitation learning
Singh et al. [48]	BDI agents	Dynamic	Plan-level	Probabilistic (online)
Faccin & Nunes [16]	BDI agents	Static	Plan-level	Supervised (predictive)
Bosello & Ricci [9]	BDI agents	Dynamic	Plan-level	RL (Q-learning)
Pulawski et al. [40]	BDI agents	Dynamic	Plan-level	RL (adversarial)
Wan et al. [56]	BDI agents	Dynamic	Plan-level	RL (Q-learning)
This thesis	**BDI/MAPF**	**Dynamic**	**Algorithm-level**	**RL (DQN)**

As Table 2.1 makes clear, existing approaches partition into two groups that are each

incomplete in a complementary sense. Works in the algorithm selection tradition [44, 24, 43, 2] operate at the correct level of abstraction, selecting among algorithmic implementations, but treat the decision as static and pre-execution, with no mechanism for revision as environmental conditions evolve. Works in the learning-enhanced BDI tradition [48, 9, 40, 56] support dynamic, runtime adaptation within the deliberation cycle, but operate exclusively at the plan-level and do not address the problem of selecting among algorithmic implementations of means-end reasoning. While Damani et al. [12] operates in a dynamic, algorithm-level regime, it bypasses explicit algorithm selection entirely, collapsing the decision into an end-to-end learned policy with no interpretable selection structure. This thesis occupies the vacant cell in this design space: it proposes a dynamic, algorithm-level selection mechanism embedded within a BDI deliberation cycle, learned via a Deep Q-Network.

This thesis addresses the gap by proposing an extended BDI architecture in which a reinforcement learning meta-reasoner, implemented as a Deep Q-Network, is embedded within the means-end reasoning component and tasked with selecting among alternative planning algorithms based on the agent's current belief-intention state. Algorithm selection is formalised as a sequential Markov Decision Process, extending Rice's classical static framework [44] to dynamic settings where the problem instance evolves during execution. The following chapter begins to lay the empirical groundwork for this contribution by conducting a comprehensive analysis of algorithm performance variability in pathfinding environments, demonstrating the extent and structure of the performance differences that motivate the meta-reasoning approach.

Chapter 3

Understanding Algorithm Performance Variability

3.1 Introduction

Autonomous agents deployed in real-world applications often face environments where no single algorithmic approach consistently outperforms all alternatives. For instance, in pathfinding, classical A* [18] performs optimally in static environments, however there is a wide range of other alternatives A* extensions, such as D*, D* Lite, and LPA* [51, 26, 28], that have been proposed for other settings, and thus excel in different scenarios. This raises our **RQ1**: to what extent does algorithm performance depend on environmental characteristics? This chapter addresses this question through a comprehensive empirical analysis of six heuristic search algorithms namely D*, D* Lite, LPA*, LRTA*, RTAA*, and ARA* (described in Section 2.3), across systematically varied grid-based pathfinding scenarios. We measure their performance along three dimensions (path length, memory consumption, solving time) while varying three environmental features (grid size, obstacle density, start-goal distance), generating various controlled problem instances. Several prior studies have compared pathfinding algorithms in controlled settings [32, 60, 39], however, they mostly used only one or two features in isolation. Our results show that algorithm performance is highly dependent on environmental characteristics. Grid size strongly correlates with memory consumption and solving time; start-goal distance significantly impacts path length and computational effort; obstacle density shows more modest effects on certain algorithms. These findings motivate the need for an adaptive algorithm selection that is based on the environment. The structured variability documented here forms the empirical

foundation for Chapter 4, which investigates whether these performance patterns are sufficiently regular to support data-driven algorithm selection.

The remainder of this chapter is organized as follows. Section 3.2 describes the algorithms and experimental domain. Section 3.3 defines the environmental features and their discretization. Section 3.4 presents experimental procedures, results, and discussion.

3.2 Algorithms Under Study

Our study investigates the performance of six heuristic search algorithms, namely D*, D* Lite, LPA*, LRTA*, RTAA*, and ARA* introduced in Chapter 2 (Section 2.3). The selected algorithms span the principal classes of heuristic search, including incremental replanning, real-time search, and anytime planning, thereby covering a broad range of computational trade-offs relevant to pathfinding.

In this study, we employ the Euclidean distance heuristic for all algorithms to guide their search. For LRTA* and RTAA*, we set a fixed lookahead of 250 nodes per iteration, a mid-range setting that balances planning depth against computational cost and is sufficient to capture meaningful local structure while maintaining real-time performance. For ARA*, we used a heuristic weight of 2.5 to balance the trade-off between search speed and optimality, following the initial value established in the original ARA* paper [**?**]. The algorithms differ fundamentally in scope: incremental algorithms (D*, D* Lite, and LPA*) maintain global search information that can be reused across replanning episodes, and ARA* similarly maintains a global search frontier while progressively refining its solution. In contrast, LRTA* and RTAA* restrict computation to bounded local lookahead windows, updating heuristic values online and bounding resource use independently of path length. This global-versus-local divide is the primary lens for interpreting the results in Section 3.4.

Typically, these algorithms are implemented in grid-based environments to simulate real-world scenarios, particularly in autonomous navigation and robotics [14]. Given their simplicity, control ease, and being commonly used in path-planning tasks in the research community, we decided to use grid-based environments. In our grid-based environment, white cells represent traversable cells, while black cells are non-traversable ones, i.e., obstacles. The agent can move in eight possible directions, with a cost equal to 1 for horizontal and vertical moves, and $\sqrt{2}$ for diagonal moves.

3.3 Environment Characterization

This study extends our previous work [23] by investigating how different grid-based features affect the performance of various search algorithms, considering the same environment features: grid size (height and width), obstacle density, and start-goal distance. We selected these features because they capture three complementary sources of planning complexity: grid size determines the size of the search space, obstacle density influences traversability, and start-goal distance determines the expected planning horizon. Together, they characterise both the structural and geometric properties of the environment that directly influence heuristic search behaviour.

Using three levels per feature provides the minimum granularity required to capture low, medium, and high values of each environmental characteristic while keeping the experimental design computationally manageable. This enables the observation of broad performance trends across levels while avoiding the combinatorial cost of finer discretisation. Varying all three features jointly, rather than in isolation, yields a fully factorial set of 27 problem configurations, allowing the combined influence of multiple environmental characteristics to be examined.

- **Grid size:** To assess the impact of the environment size, we used three grid sizes: 50×50 (small), 175×175 (medium), and 300×300 (large). These sizes span small, medium, and large environments representative of commonly used robotics and navigation benchmarks [52].

- **Obstacle density:** Obstacle density is defined as the percentage of grid cells occupied by non-traversable obstacles. We evaluated three levels of obstacle density: 10% (sparse), 25% (dense), and 40% (very dense). These densities reflect a gradient from environments with few obstacles to those in which navigation is greatly challenged by the presence of many obstacles.

- **Start-goal distance:** To make the categories comparable across different grid sizes, start-goal distance is expressed relative to the maximum possible diagonal distance of the grid, $(n-1)\sqrt{2}$ for a grid of size $(n \times n)$. Distances up to 20% of this value are classified as "very close", distances between 20% and 60% as "close", and distances above 60% as "far".

3.4 Experimental Design and Results

To evaluate how these environmental features influence search performance, we measured both the effectiveness of each algorithm (the quality of the solution it produces) and its efficiency (the computational resources it consumes) using metrics commonly adopted in path-planning evaluations [32, 60, 39, 25]. The selected metrics are:

- **Path cost:** This metric measures the path length (i.e., the number of executed actions) from the start to the goal state, indicating the quality of the solution.
- **Memory consumption:** It measures the required amount of memory for the algorithm to find a solution, measured in kilobytes (KB).
- **Solving time:** Represents the total time an algorithm takes to find a solution, measured in milliseconds (ms).

Together, these metrics provide complementary perspectives on performance relevant to real-world deployment: path cost reflects solution quality (critical for minimizing energy consumption in robotics), solving time measures computational efficiency (essential for real-time systems), and memory consumption determines scalability (important for resource-constrained devices). Considering all three avoids favouring algorithms that optimise only a single performance dimension.

3.4.1 Experimental Procedure

For each combination of grid size, obstacle density, and start-goal distance, we generated ten independent random grid instances, resulting in 27 environmental configurations and 270 benchmark problems. Within a configuration, the start and goal positions are held fixed and only the obstacle layout is resampled across the ten instances; this isolates the effect of obstacle placement and ensures that the results are not skewed by the arbitrary distribution of obstacles in the grid. Each generated instance was verified to be solvable by running A* on it; layouts admitting no path between start and goal were discarded and regenerated. Each algorithm was executed five times on every instance to account for runtime variability, particularly in solving time, thereby confirming the consistency and reliability of the algorithm's performance.

The experiment was conducted on 3.30GHz 27 Intel i9 cores, equipped with 250Gb of RAM and running Ubuntu Linux 22.04 to ensure that our results accurately reflect the algorithm capabilities, free from hardware limitations.

3.4.2 Experimental results

This section evaluates how the three environmental features introduced in Section 3.3 influence algorithm performance in terms of path cost, memory consumption, and solving time.

- **Effect of grid size on algorithm performance:** Across all algorithms, grid size correlates moderately to strongly with mean path length (0.63-0.66) and mean solving time (e.g., 0.87 for RTAA*), and strongly with allocated memory (up to 0.97 for D*, LRTA*, and RTAA*), as shown in the correlation matrices in Figures 3.6a-3.6f. Figure 3.1(a) confirms this for path length: most algorithms follow near-similar paths regardless of grid size, with mean path length increasing consistently as grid size grows.

  Memory consumption, shown in Figure 3.1(b), escalates with grid size but diverges sharply across algorithms: D* Lite consumes the least memory at all grid sizes except below 75, where ARA* is marginally more efficient. This divergence stems from the algorithms' underlying data structures. D* maintains back-pointers for every expanded node to support path replanning, a strategy that grows increasingly costly as the state space expands. RTAA* and LRTA*, by contrast, operate within a bounded lookahead of 250 nodes per iteration, which caps per-step memory allocation regardless of grid size, at the cost of global optimality.

  Solving time follows a similar pattern, shown in the same figure: it increases with grid size, but the magnitude of this increase varies by algorithm, with RTAA* consistently fastest, followed by ARA* and D* Lite.

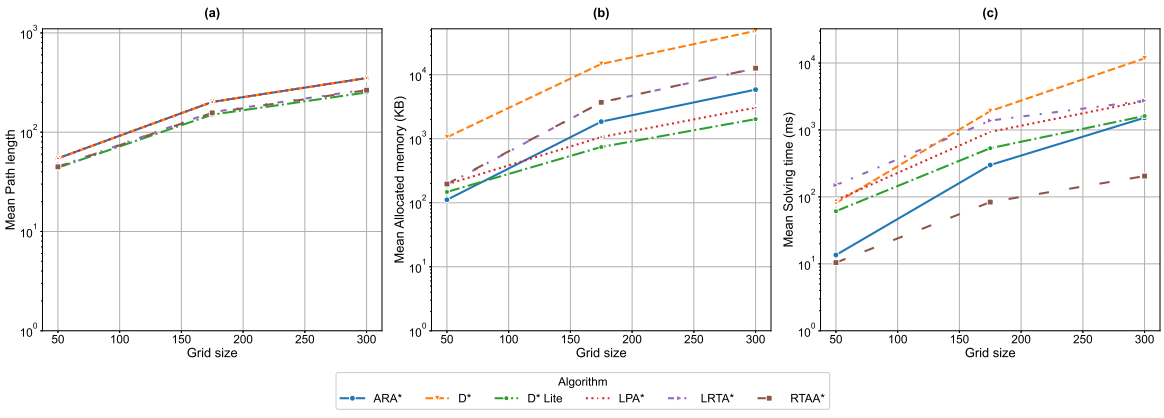


Figure 3.1: Performance metrics vs grid size

- **Effect of start-goal distance on algorithm performance:** By maintaining a consistent obstacle density across three grid sizes and evaluating three start-goal distances (very close, close, and far) at each size, we assessed how start-goal distance affects algorithm performance. Across all algorithms, start-goal distance correlates moderately to strongly with all three metrics (Figures 3.6a-3.6f): path length reaches a near-perfect correlation of 1.00 for D* Lite and RTAA*, and 0.99 for LRTA*; solving time is also strongly correlated (0.92 for D* Lite, 0.91 for RTAA*); memory correlation is more variable (0.91 for ARA*, 0.65 for RTAA*).

  Figure 3.2 confirms that path length increases with start-goal distance across all grid sizes, as expected. More notably, algorithms converge toward similar path lengths as distance increases: this near-perfect correlation reflects a topological inevitability, since greater start-goal separation geometrically constrains the minimum traversal cost regardless of algorithm, reinforced by the shared Euclidean heuristic guiding all algorithms toward comparable routes once the search space is large and obstacle density is moderate. The discriminating factor therefore shifts from solution quality to computational cost, explaining why solving time and memory, rather than path length, differentiate algorithm behaviour at longer distances.

  Figure 3.3 shows that start-goal distance affects memory requirements differently across algorithms. ARA* is the most affected, requiring substantially more memory as distance increases across all grid sizes, while D*, RTAA*, and LRTA* show only slight increases; LPA* and D* Lite are more affected specifically at larger grid sizes (175 and 300) than at grid size 50. ARA*'s sharply increasing memory demand reflects its anytime search strategy: it progressively refines an initial suboptimal solution by re-expanding nodes under decreasing heuristic weights, so longer trajectories require more refinement passes and proportionally larger open and closed sets. D*, RTAA*, and LRTA* do not maintain a global closed set of comparable breadth, so their memory growth is moderated either by incremental update propagation (D*) or by the fixed lookahead window (RTAA*, LRTA*), which truncates the search horizon independently of path length. D* Lite remains the most memory-efficient algorithm at longer distances despite its incremental replanning overhead, since its LPA*-derived architecture updates only the affected portion of the search graph rather than re-evaluating the full trajectory. Across all grid sizes, ARA* consumes the least memory at short distances, D* Lite becomes the most memory-efficient as distance grows, and D* consistently requires

the most memory of all algorithms.

Figure 3.4 shows that solving time diverges across algorithms as start-goal distance increases: ARA* recorded the fastest solving times at short distances, while RTAA* recorded the fastest solving times as distance increased; D* was the slowest algorithm at all distances and grid sizes, except at grid size 50 at longer distances, where LRTA* became the slowest. ARA*'s advantage at short distances arises from the early termination afforded by its inflated heuristic (weight 2.5): when start and goal are proximate, the inflated heuristic directs search aggressively toward the goal, and an initial solution is found rapidly without costly refinement. As distance grows, the number of required refinement iterations increases substantially, eroding this advantage. RTAA*'s advantage at longer distances stems from its bounded lookahead of 250 nodes: rather than planning a complete path, it commits to locally optimal moves within that window and updates heuristic values online, distributing computational effort across many small increments rather than incurring a large upfront planning cost.

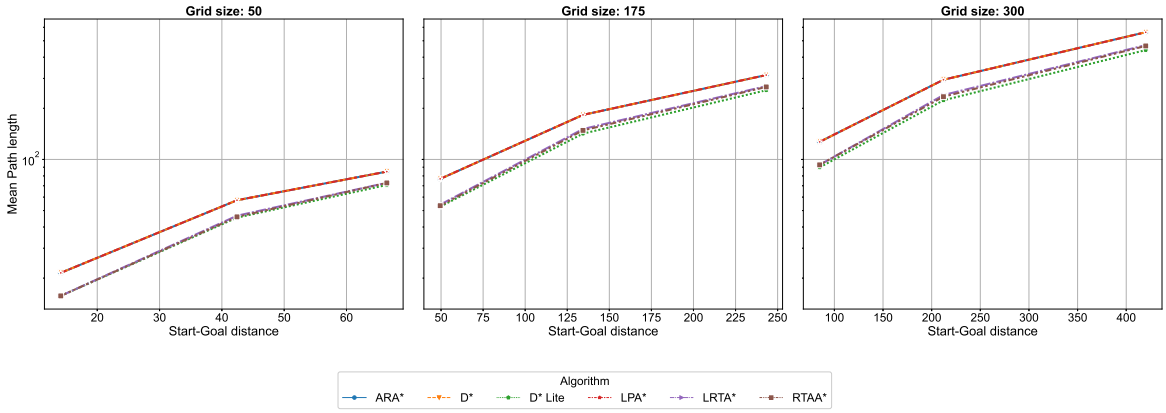


Figure 3.2: Mean path length by start-goal distance

- **Effect of obstacle density on algorithm performance:** To evaluate the effect of obstacle density, we held grid size and start-goal distance constant while varying density. In contrast to grid size and start-goal distance, obstacle density shows a negligible to weak correlation with algorithm performance (e.g., 0.07 for RTAA* path length, 0.00 for RTAA* memory consumption, -0.02 for D* memory consumption), reflected in the cool grey shades of the correlation matrices in Figures 3.6a-3.6f.

  In addition to the heatmaps, figure 3.5 details this relationship per metric. Path length (see plot (a) on the left) is only slightly affected by density for D* Lite,

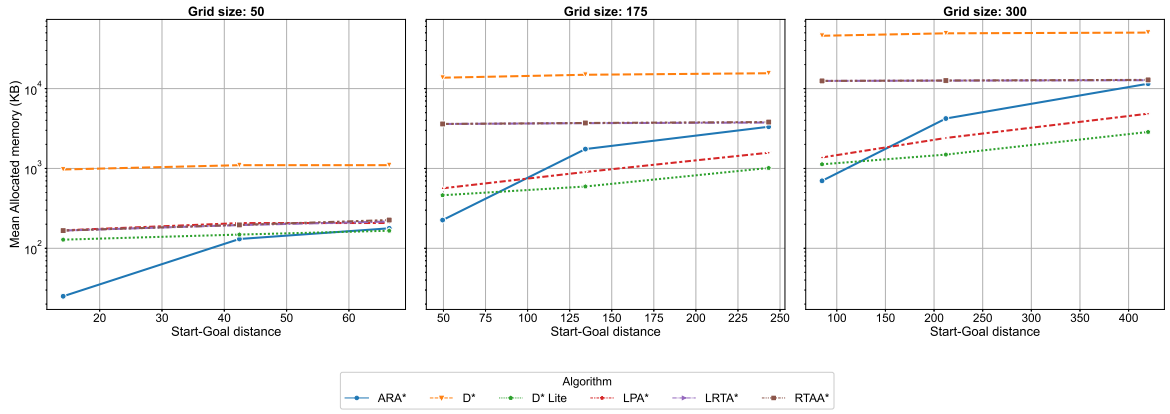


Figure 3.3: Mean allocated memory (KB) by start-goal distance

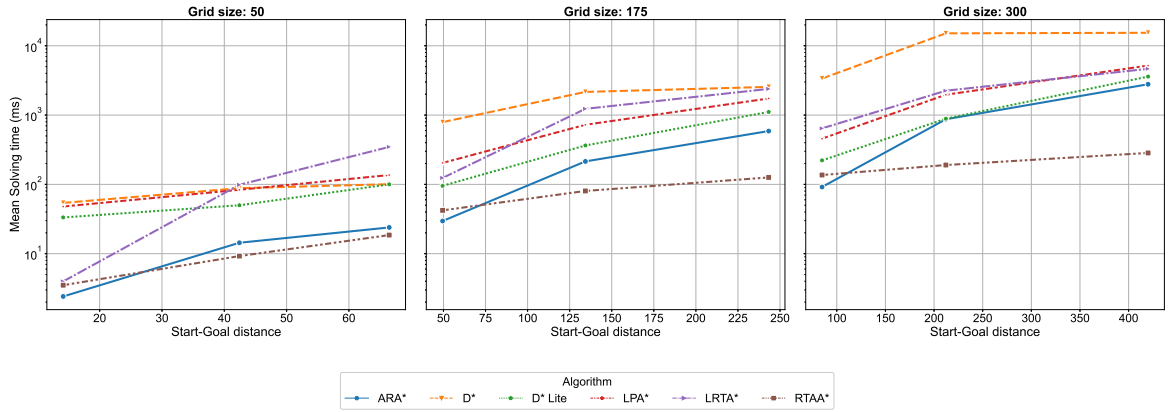


Figure 3.4: Mean solving time (ms) by start-goal distance

RTAA*, and LRTA*, but ARA*, D*, and LPA* show a marked spike at 40% density. Similarly, when it comes to observing the memory consumption (plot (b)) is similarly stable for D*, RTAA*, LRTA*, and ARA*, with D* Lite and LPA* requiring somewhat more memory at higher densities; D* Lite remains the most memory-efficient algorithm overall, and D* the least. Solving time (plot (c)) is largely unaffected by density for all algorithms, with RTAA* consistently fastest and D* consistently slowest.

Overall, the results demonstrate that varying the obstacle density does not affect the mean solving time of the algorithms much, and they all tend to have a stable performance.

This general insensitivity to obstacle density is likely explained by the uniform random obstacle distribution used to generate the environments: randomly placed obstacles rarely form the contiguous barriers, such as corridors or bottlenecks, that force substantial detours. The path-length spikes at 40% density for ARA*, D*, and LPA* are consistent with this explanation, since these are precisely the algorithms that maintain global search structures, where denser grids increase the number of node re-expansions required to update the global cost estimate, which is an overhead absent in the bounded real-time algorithms.

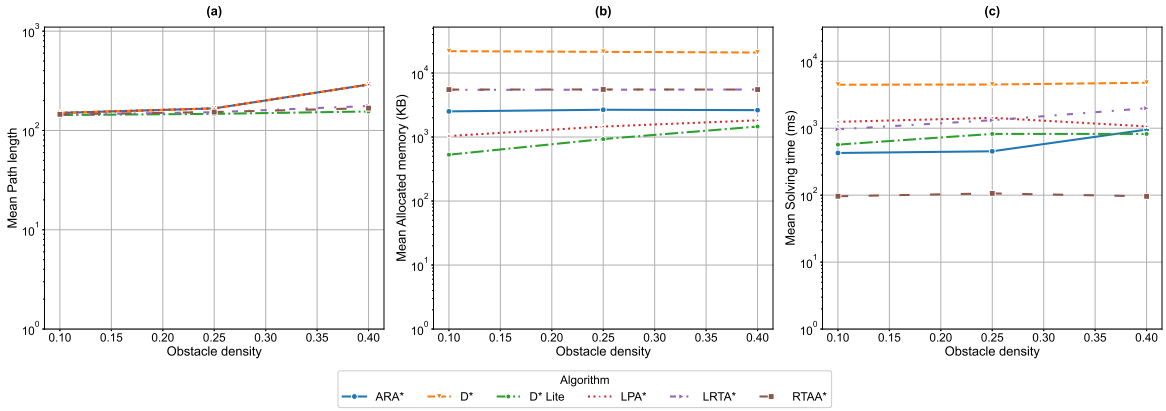
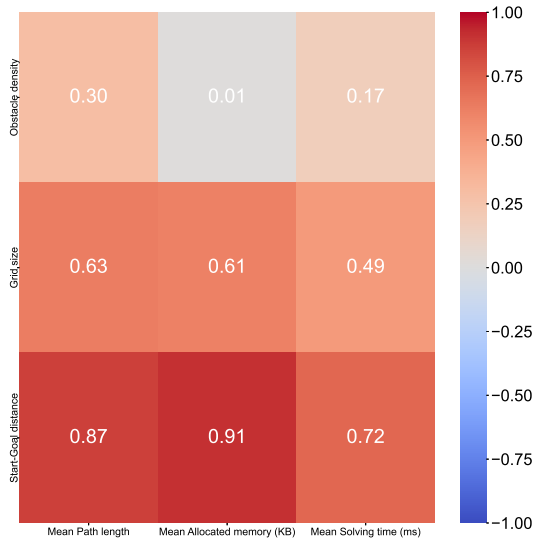


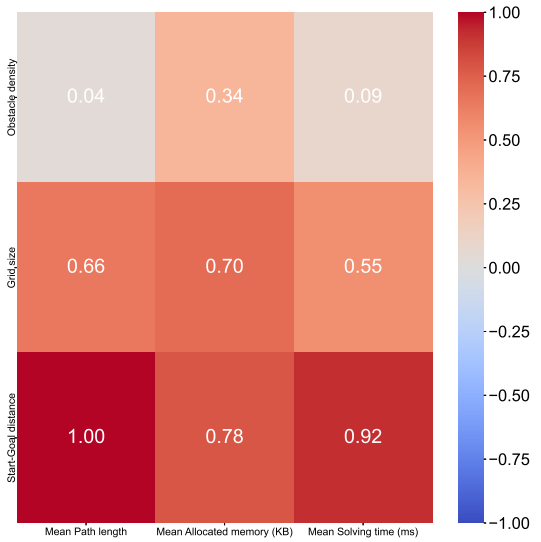
Figure 3.5: Performance metrics vs obstacle density

3.4.3 Discussion of the Results

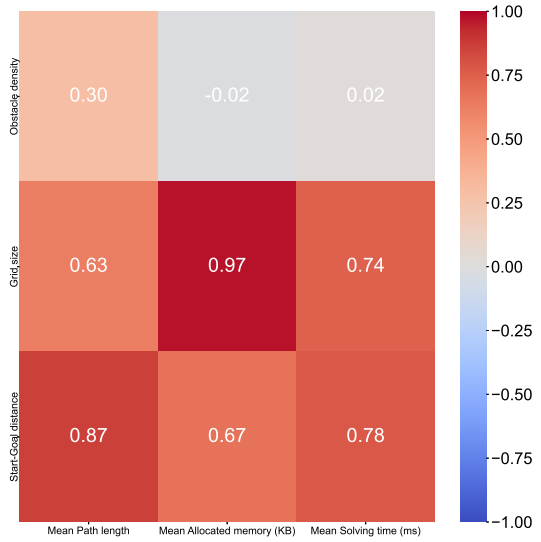
The results across all three environmental features can be understood through a single structural distinction: whether an algorithm maintains a **global, consistent cost structure** across the entire search space, or operates through **bounded, local search**



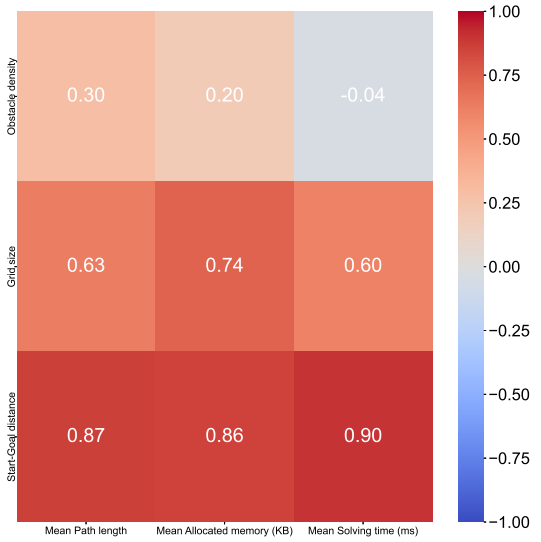
(a) ARA* correlation matrix



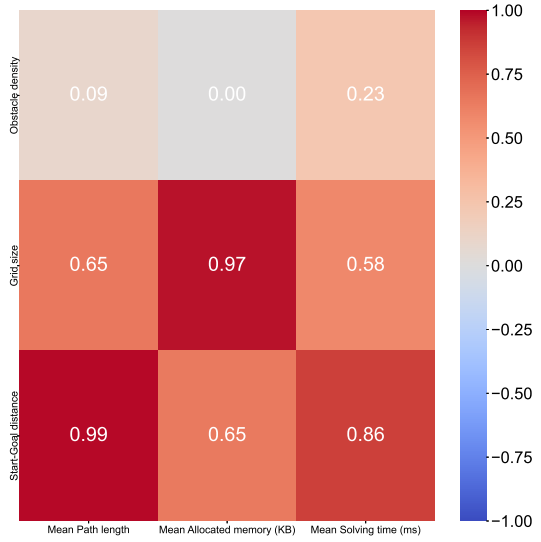
(b) D* Lite correlation matrix



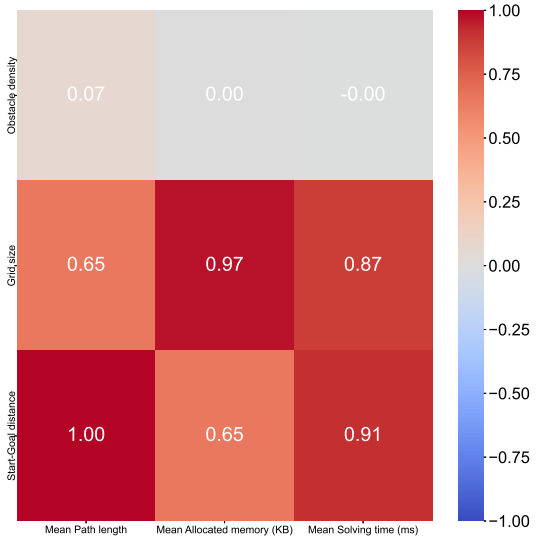
(c) D* correlation matrix



(d) LPA* correlation matrix



(e) LRTA* correlation matrix



(f) RTAA* correlation matrix

Figure 3.6: Correlation matrices

steps with online heuristic updates. Algorithms in the first category (i.e., D*, D* Lite, LPA*, and ARA*) carry memory and replanning costs that scale with the reachable state space, making them sensitive to any environmental factor that expands the effective search frontier, particularly grid size and, at high densities, obstacle configuration. Algorithms in the second category (i.e., RTAA* and LRTA*) bound their per-step computational commitment to a fixed lookahead window, which decouples their resource consumption from global state space size and explains their relative insensitivity to both grid size and obstacle density, at the cost of global optimality. This taxonomy is not incidental to the observed performance patterns: it *determines* them, and it constitutes the structural basis on which any principled algorithm selection strategy, including the learned meta-reasoner introduced in Chapter 5, must be grounded.

We observed that the grid size is a significant grid feature that influences both path length and solving time across all algorithms. This is a clear indication that larger grids naturally demand more computational resources and more time to find a solution due to the increase in nodes and possible paths. The results also show that sensitivity to grid size varies by algorithm and metric: D* exhibits the sharpest memory growth due to its global back-pointer structure, while RTAA* and LRTA* show sensitivity primarily in solving time, as longer paths require more bounded-lookahead iterations to complete, even though their per-step memory allocation remains fixed.

The influence of start-goal distance confirms that computational effort is driven not only by the size of the search space but also by the amount of traversal required to reach the goal. As trajectory length increases, computational cost becomes the primary differentiating factor between algorithms, whereas solution quality converges because all algorithms are guided by the same heuristic. Consistent with these structural differences, ARA* is preferable for short trajectories, whereas RTAA* becomes increasingly advantageous as planning horizons grow, as discussed in Section 3.4.2. When memory consumption becomes a priority, D* Lite is the algorithm best suited to generating the shortest paths using the least memory in most cases. In contrast to the first two grid features, obstacle density had a minimal impact on the performance of the algorithms. When increased, it only affected the path length of some algorithms, which are D*, ARA*, and LPA*. This modest effect, attributable to the uniform random obstacle distribution as discussed above, may also reflect a known limitation with respect to real-world navigation benchmarks [53]. Whether denser or more structured obstacle configurations would produce stronger effects on algorithm performance remains an open question.

Overall, the experiments demonstrate that no single algorithm dominates across

all environments. D* incurs substantial memory overhead as the search space expands, whereas D* Lite consistently offers the best memory efficiency. RTAA* provides the best runtime performance in larger environments and over longer planning horizons, while ARA* remains competitive in smaller environments with short start-goal distances. These observations provide practical guidelines for selecting planning algorithms according to environmental characteristics rather than relying on a single fixed strategy.

Collectively, these findings answer RQ1 by demonstrating that algorithm performance varies systematically with identifiable environmental characteristics rather than remaining consistent across problem instances. This variability provides the empirical basis for the learning-based selection developed in the following chapters.

3.5 Summary

This chapter analysed how different environmental features influence the performance of six heuristic search algorithms across a range of varied grid-based pathfinding scenarios. The results demonstrate the algorithm's performance is strongly shaped by the structure of the environment itself.

Among the selected grid features, grid size emerged as the most influencing feature on both memory consumption and solving time across all algorithms. In contrast, start-goal distance had a similar significant impact on path length and computational effort, with ARA* proving most effective for short distances and RTAA* excelling at longer ones, while D* Lite offered the best memory efficiency across varying distances. Obstacle density, on the other hand, influenced performance to a minor extent. Only D*, ARA*, and LPA* showed noticeable increases in path length at higher densities, while the remaining algorithms were comparatively unaffected. This suggests that global spatial properties, such as grid size and start-goal separation, are more informative indicators of algorithm behavior than obstacle density in isolation. Underlying these patterns is a structural distinction between algorithms that maintain global, consistent cost representations such as D*, D* Lite, LPA*, and ARA*, and those that operate through bounded local search with online heuristic updates such as RTAA* and LRTA*; this distinction is the primary determinant of how each algorithm responds to changes in grid size, start-goal distance, and obstacle density.

Overall, these results support the central argument of this thesis: algorithm performance depends in a systematic way on identifiable environmental features, and no algorithm dominates across all scenarios. This variation provides both the motivation and the foundation for the inquiry that follows: the structured variability documented here

is a necessary precondition for any learning-based selection strategy. Chapter 4 investigates whether this structured variability is also sufficient to support reliable learning-based algorithm selection and, by exposing the limits of static prediction, motivates the adaptive architecture developed in Chapter 5.

Chapter 4

Automating Algorithm Selection via Machine Learning

4.1 Introduction

Our analysis in Chapter 3 established that algorithm performance varies systematically with manually defined grid-based characteristics such as grid size, obstacle density, and start-goal distance, and that no single pathfinding algorithm consistently outperforms others across all scenarios. This variability motivates the next question: can the most suitable algorithm be predicted from characteristics of the environment before planning begins?

In this chapter, we further investigate whether this performance variability can be exploited through data-driven prediction, addressing **RQ2**. We model algorithm selection as a supervised classification task: we train classification models to predict the most suitable search algorithm from the environmental features of a problem instance, with separate models trained for each optimization objective: minimizing path length, solving time, and memory consumption.

We proceed our investigation as follows. We first describe the data collection process, in which pathfinding algorithms are run on MAPF benchmark scenarios to generate labelled training examples. We then consider two types of features: grid-based spatial characteristics and CNN-derived features capturing visual structure. Following data collection, we train multiple Random Forest classifiers using different feature combinations and evaluate their prediction accuracy on held-out test instances.

Evaluating our results, we demonstrate that environmental features can effectively predict optimal algorithm choices for specific performance objectives, thereby validat-

ing the feasibility of learning-based algorithm selection. However, this approach operates under a fundamental assumption: that environmental conditions remain static throughout problem-solving. As we will show, this assumption limits the applicability of supervised prediction to dynamic settings where the environment conditions evolve during execution.

4.2 Preparation and Labelling of the Datasets: Running Search Algorithms Over the MAPF Benchmark Dataset

To train machine learning models capable of predicting the most efficient search algorithm for a pathfinding scenario, we required a dataset with greater environmental diversity than the controlled grids of Chapter 3. Building on our prior work [24], we therefore extended our experiments to the well-known multi-agent pathfinding (MAPF) benchmarks [52], which are widely used in the pathfinding literature and, unlike our earlier synthetic grids, vary in map topology, size, and obstacle distribution. We evaluate the same six heuristic search algorithms studied in Chapter 3 (Section 3.2) that are D*, D* Lite, LPA*, LRTA*, RTAA*, and ARA* using the same fixed parameter settings established and justified there: a lookahead of 250 nodes for LRTA* and RTAA*, and a heuristic weight of 2.5 for ARA*.

We used every MAPF map except `orz900d`, which we excluded due to its excessive computational cost (at 656×1491 cells, it is by far the largest map in the suite); the complete list of the 32 maps used, with their categories and dimensions, is provided in Appendix A. For each remaining map, we generated 25 random scenarios (i.e., 25 distinct start-goal position pairs), yielding a total of 800 problem instances.

4.3 Feature Extraction and Data Labeling

Preparing our dataset involved two steps: extracting features from the MAPF benchmark maps, and labelling each instance according to search algorithm performance.

4.3.1 Input Feature Extraction

We use two different methods to characterize the MAPF maps, used to generate two different training sets:

- **Grid-based features:** consists of the manually defined grid-based features introduced in Chapter 3 (Section 3.3): map size (height and width), obstacle density, and the Euclidean distance between the start and goal positions, computed directly from the scenario and map files available on the MAPF benchmarks website.

- **CNN-extracted features:** Scalar grid features summarise global spatial statistics but cannot encode the internal structure of a map. For example, two maps with identical obstacle density can differ substantially in layout (i.e., one may contain open corridors with localised clusters, the other uniformly distributed obstacles) and such structural differences affect pathfinding algorithm performance [43, 37]. Moreover, treating the map as a two-dimensional image and applying a CNN is a principled approach to capturing these spatial patterns, with direct precedent in the MAPF algorithm selection literature [43]. We use the VGG16 model [47], whose hierarchically stacked convolutional and pooling layers extract features at increasing levels of abstraction, yielding a 512-dimensional representation of map structure that scalar features alone cannot provide.

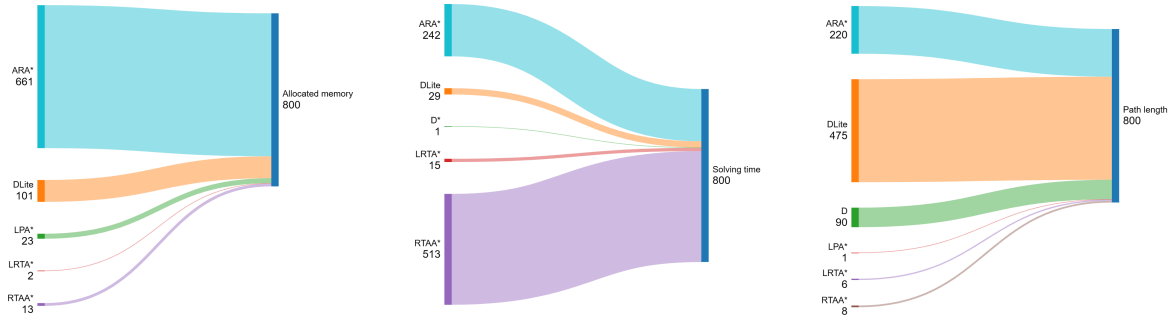
While the first approach considers simple spatial characteristics of the maps, the second method extracts high-level features within the images of MAPF maps, potentially capturing complex patterns within the maps that are not evident through grid features only.

4.3.2 Data Labelling

Once we obtained the features, we ran and recorded the search algorithm's performance for each instance in terms of the same three metrics defined in Chapter 3 (Section 3.4): solving time, memory consumption, and path length. Based on these metrics, we labelled each instance with the best-performing algorithm according to each criterion. This process resulted in three different datasets, with each highlighting the best algorithm for a particular performance criterion, visualised as Sankey diagrams in Figure 4.1. The bands in the diagrams reflect the frequency with which each algorithm was labelled as the best performer given an instance and according to a performance metric. The diagrams show that ARA* has the highest frequency for minimal memory usage (4.1a), RTAA* for solving time (4.1b), and D* Lite for path length (4.1c).However, the presence of other algorithms in each criterion confirms that no single algorithm is uniformly best, consistent with the variability established in Chapter 3.

During the labelling process, we encountered many instances where multiple algo-

rithms were equally optimal for minimal path length criterion. To handle the latter, we assigned multiple labels to the instance, allowing the model to consider all possible best algorithms rather than only one best algorithm. However, for the criteria of minimal solving time and minimal memory usage, the labelling process was more straightforward. Each scenario was linked to a single best-performing algorithm, resulting in a single-label classification setup.



(a) Data labelled with the best algorithm for minimal memory usage

(b) Data labelled with the best algorithm for minimal solving time

(c) Data labelled with the best algorithm for minimal path length

Figure 4.1: Labelled data sets

4.3.3 Preparing the Labelled Dataset

After extracting features and labelling the data, we prepared the final datasets to be used as inputs for training machine learning models. Each model is trained to predict the best-performing algorithm for each criterion: minimal solving time, minimal memory consumption, and minimal path length. For each criterion, we trained 3 separate models, each taking a different set of input data:

- **Grid Features Alone:** This model is trained exclusively using the grid-based features that are the map size including the width and the height, the obstacle density, and the distance between the start and the goal positions.
- **CNN-Extracted Features Alone:** The second model is trained solely on features extracted from the map images using the VGG16 model.
- **Combined Features:** The third model is trained on a dataset that combines both sets of grid features and CNN-Extracted features.

For each selection criterion, the labelled dataset was partitioned into training and test sets using a 75/25 split with a fixed random seed to ensure reproducibility. For the single-label criteria (solving time and memory consumption), the split was stratified on the target label to preserve class proportions across both sets; for the multi-label path-length criterion, a plain random split was used, since stratification is not directly applicable to multi-label targets. The split was performed at the level of individual problem instances; scenarios drawn from the same map may therefore appear in both sets, and the reported results measure generalisation to unseen start-goal configurations rather than to entirely unseen maps.

For the multi-label path-length models, this corresponds to the strict subset (exact-match) definition of accuracy, under which a prediction is counted as correct only if the complete set of predicted labels coincides with the ground truth; partially correct predictions receive no credit.

4.4 Models Training

We have used the Random Forest (RF) algorithm [10] to predict the best-performing algorithm based on different selected criteria. RF is well-suited to our data: first, it supports multi-label classification that is required in the minimal path length criterion, where multiple algorithms may be jointly optimal. Second, it handles class imbalance through instance-level class weighting, addressing the distributional skew visible in Figure 4.1. Third, it handles mixed scalar and CNN-derived features without preprocessing (i.e., low-dimensional scalar grid features with high-dimensional CNN-extracted representations) without the need for feature normalization or architectural redesign. Fourth, it provides feature importance scores to interpret which environmental characteristics drive algorithm selection, which is crucial in this study. These properties make RF a principled and interpretable choice for this task.

The models were further fine-tuned using the GridSearchCV from the Scikit Learn framework [38] to tune the estimator's hyperparameters, including the number of trees in the forest, the maximum depth of the trees, the number of samples required to split an internal node, the number of samples required to be at a leaf node, and the class weight used to handle imbalances. This optimization step was very important to improve the model performance.

We evaluated the performance of our models using several metrics.

- Accuracy: It refers to the percentage of correct predictions that the model has

made out of the total number of predictions; it is represented in the formula 4.1.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \times 100 \tag{4.1}$$

- Hamming Loss: A metric computing the fraction of wrong labels in multi-label classification. It is useful when evaluating models in which multiple labels can be assigned to the same instance, such as in our model predicting the best algorithm based on minimising the path length. Hamming loss can be calculated as in formula 4.2.

$$\text{Hamming Loss} = \frac{1}{N}\sum_{i=1}^{N}\frac{\text{Incorrect Labels for Instance } i}{\text{Total Labels for Instance } i} \tag{4.2}$$

- ROC AUC: The Receiver Operating Characteristic, Area Under the Curve (ROC AUC) [17] is useful in assessing the model's ability to distinguish between classes. A model with a higher AUC score indicates better performance in differentiating between the best algorithms. ROC AUC can be calculated by plotting the true positive rate against the false positive rate and measuring the area under the resulting curve.

4.5 Model's Evaluation

Table 4.1 summarises the performance metrics obtained from the models predicting the best algorithm according to the criterion of minimal path length. The model taking only grid features as inputs shows an accuracy of 49% and a hamming loss of 0.2175, meaning that it correctly predicted the best algorithm 49% of the time but made a significant number of errors, as indicated by the hamming loss score. These results are similar to those obtained from the model using only CNN features, which achieved an accuracy of 48% and a hamming loss of 0.2425. However, when training the model using both feature sets, we observe higher scores, with the model correctly predicting the best algorithm 52% of the time and a reduced number of incorrect labels as the hamming loss shows a value of 0.1967.

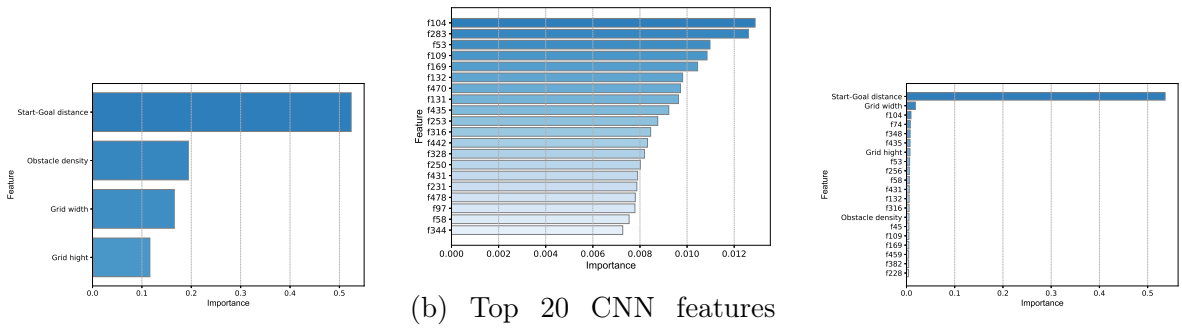
The charts 4.2a, 4.2b, and 4.2c in Figure 4.2 show the importance of the input features used to predict the best-performing algorithm for path length in each model. Start-goal distance is the top feature in both the grid-only and combined models. Using grid features only, it is followed by obstacle density, then grid width and grid height.

Table 4.1: Performance metrics summary for minimal path length

Model	Accuracy	Hamming Loss
Grid features only	49%	0.2175
CNN features only	48%	0.2425
Combined features	52%	0.1967

When both feature sets are combined, start-goal distance dominates even more strongly, with grid width emerging as the next most important grid-based feature; grid height and obstacle density, by contrast, rank considerably lower, behind several CNN-derived features.

Start-goal distance and grid width, in particular, appear to account for the accuracy improvement and reduced Hamming loss of the combined model over the single-feature-set models: both outscore every CNN-derived feature in Figure 4.2c, whereas grid height and obstacle density rank below several CNN features in that same model.



(a) Grid features only

(b) Top 20 CNN features only

(c) Combined features

Figure 4.2: Feature importance for minimal path length criterion

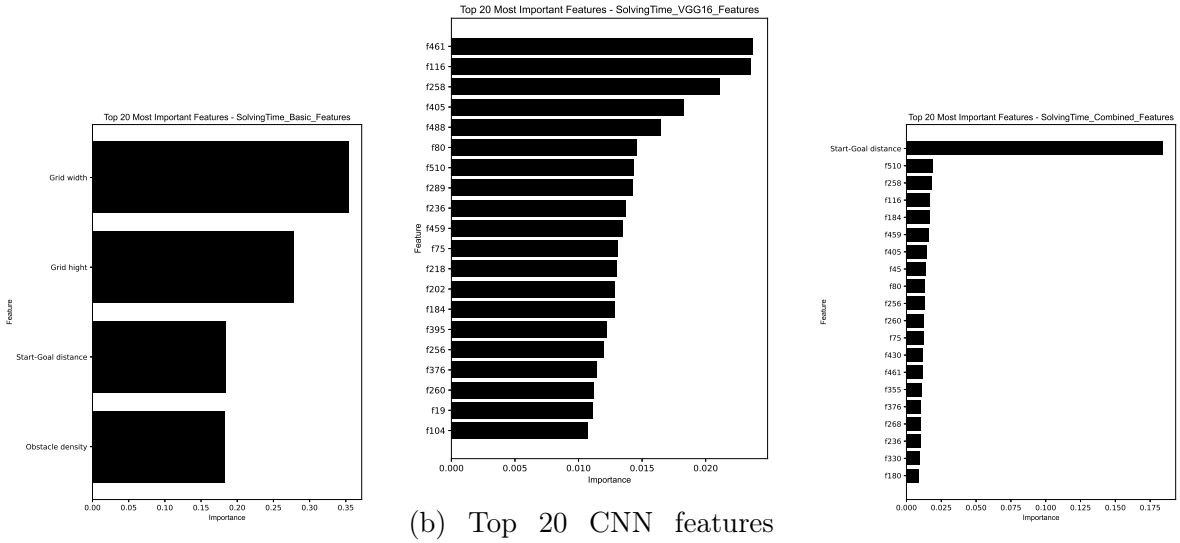
Table 4.2 highlights the performance metrics for models trained to select the best algorithm that minimizes the solving time. The first model, using only grid features, predicted correctly the best algorithm with an accuracy of 76.5%. It also had a high ROC AUC score of 0.8998, indicating a good performance in differentiating between different classes. Compared to the first model, the second model using only CNN features shows a notably lower accuracy of 68%, but a relatively close ROC AUC value of 0.8609 to the one in the first model. In contrast to the first two models, the last model trained using both feature sets had the highest accuracy of 78% and a slightly lower ROC AUC score.

Feature importance analysis in 4.3a shows that the grid width and height are the top features when using grid features alone, followed by an equal score for start-goal distance

Table 4.2: Performance metrics summary for solving time models

Model	Accuracy	ROC AUC
Grid features only	76.5%	0.8998
CNN features only	68%	0.8609
Combined features	78%	0.8775

and obstacle density. While the model that uses CNN features only 4.3b, shows diverse scores for the feature set extracted from the images, however, when both feature sets are combined 4.3c, start-goal distance re-emerges again as the highest ranked grid feature alongside the CNN-derived features. This suggests that knowing the distance between the start and goal alongside CNN features improves the selection of the best algorithm as seen in the accuracy increase in 4.2, however since the information about the start-goal distance is absent in the maps, the CNN model didn't extract any information about it.



(a) Grid features only (b) Top 20 CNN features only (c) Combined features

Figure 4.3: Feature importance for minimal solving time criterion

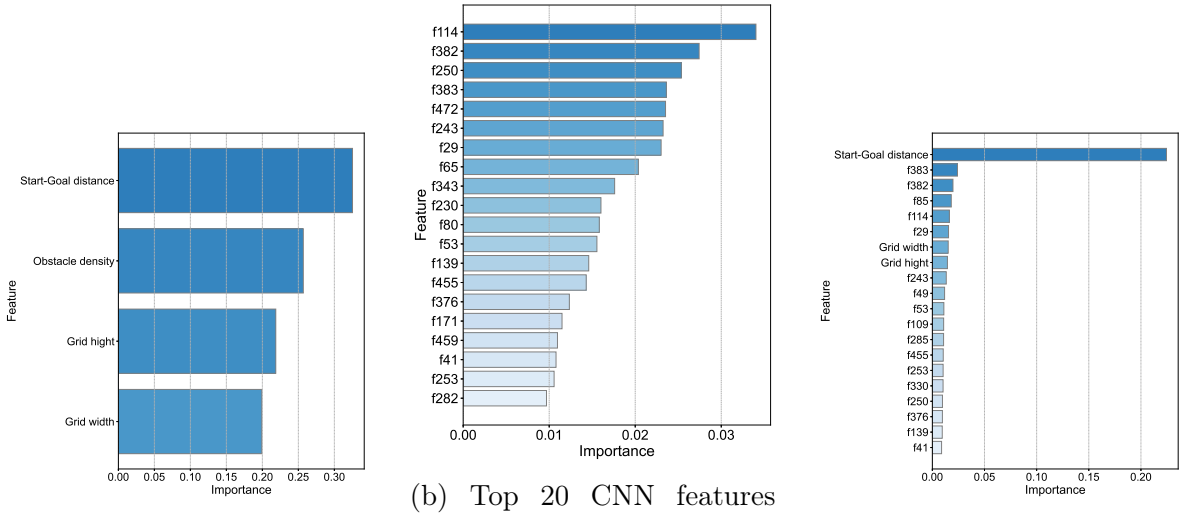
Finally, the performance metrics evaluating the trained models for memory consumption are presented in Table 4.3. The model using only grid features achieved the highest accuracy at 84%, followed closely by the combined-features model at 83%; the CNN-only model trailed at 78.5%. All three models nonetheless show good ROC AUC scores, with the highest value recorded for the combined-features model.

Analysing the feature importance for minimal memory consumption in Figure 4.4, Start-goal distance is the feature with the highest importance in selecting the best

Table 4.3: Performance metrics summary for memory consumption

Model	Accuracy	ROC AUC
Grid features only	84%	0.9342
CNN features only	78.5%	0.9328
Combined features	83%	0.9513

algorithm followed by the obstacle density and the grid height and width with small differences for the model using grid features only (4.4a).



(a) Grid features only

(b) Top 20 CNN features only

(c) Combined features

Figure 4.4: Feature importance for minimal memory consumption criterion

Similar to the previous criterion, the model that uses CNN features only (4.4b) demonstrates various scores for the feature set extracted from the images, however, when combining both feature sets (4.4c); start-goal distance merges again as the highest-scored feature followed with only CNN features to select the best algorithm for a minimal memory usage criterion.

4.6 Discussion of the Results

The results suggest that our machine learning models succeeded in predicting the best search algorithm across a variety of pathfinding scenarios. For path length and solving time, the models using both feature sets (i.e., grid features and CNN features) outperform single-feature-set models; for memory consumption, however, the grid-only model achieves marginally higher accuracy (84% vs. 83%), though the combined model retains

the highest ROC AUC. This indicates that both spatial features, such as the start-goal distance, and high-level features of the map images provide a more comprehensive understanding of the environment, leading to a more accurate algorithm selection in most cases.

Although the model using combined features achieved a higher accuracy of 52% for the path length criterion, this remains a relatively modest result that warrants critical examination. Two distinct explanations must be considered and carefully separated. The first is inherent algorithmic ambiguity: since the algorithms share the same heuristic, uniform cost model, and movement directions, they frequently recover paths of equal cost on the same scenario despite potentially exploring the search space differently, resulting in a substantial proportion of instances admitting multiple equally optimal labels, so that any single label assigned to such instances is arbitrary among several valid choices. This introduces a structural ceiling on achievable accuracy that no feature set can overcome, and it is corroborated by the comparatively low Hamming loss of 0.1967, which indicates that incorrect predictions tend to be near-optimal rather than wholly misclassified.
The second explanation is feature limitation: path length is sensitive to fine-grained topological properties of the map, such as local connectivity and corridor structure, which are not fully captured by either the three scalar grid features or the globally pooled VGG16 representations.
These two explanations are not mutually exclusive, but their relative contributions have different implications for future work. If ambiguity is the dominant factor, the ceiling on accuracy is fundamentally a property of the task and would persist even with richer features; the appropriate response is then to reformulate the problem, for instance through soft or multi-label objectives. If feature limitation is the dominant factor, the ceiling is instead a limitation of the current features and could in principle be raised by extracting more discriminative topological descriptors. On current evidence, both factors are plausibly operative, and disentangling their contributions remains an open question. The results also confirm that start-goal distance is the most influential feature, which likely accounts for the accuracy improvement of the combined model over the CNN-only model.

In the case of solving time, the model using only grid features performed nearly the same as the model using the combined features with only a difference of 1.5% in accuracy. Additionally, the ROC AUC scores for both models were slightly different. This implies that grid features that are start-goal distance, grid size (i.e., width and height), and obstacle density are probably strong features for solving time efficiency,

even though only start-goal distance appears as the most important feature in the combined model, as shown in Figure 4.3c.

Among all criteria, the model for memory consumption showed the highest accuracy, with the model using the grid features only achieving 84% accuracy; and 83% accuracy for the one using both feature sets. This indicates that the grid features have a direct relation with selecting the best algorithm for the memory consumption criterion, which can be seen in the feature importance chart in 4.4c. Furthermore, the scores obtained from the ROC AUC across all models indicate that they were effective in differentiating between the classes even in the presence of imbalanced datasets.

Analysing the feature importance across all models using grid features shows that start-goal distance followed by obstacle density are the most significant factors in determining the best algorithm for minimal path length, as well as for memory consumption. Whereas for solving time the grid width and height are the most crucial features.

Although models trained using CNN-extracted features performed less effectively compared to other models, they exhibit diversity in the relative importance of the extracted map features. The underperformance of CNN-only models is likely attributable to the loss of scalar spatial information, such as map dimensions and start-goal distance, during the image-based feature extraction process, which explains why grid dimension features appear among the top 20 important features even in the combined model. More broadly, the consistent dominance of grid-based features across all three criteria indicates that the compact spatial descriptors already encode the most predictive signal available. CNN features are therefore best characterised as complementary rather than necessary: they provide supplementary structural information that can marginally improve discriminability in some criteria, but models trained on grid features alone sacrifice little predictive power. This has a practical implication for deployment: in resource-constrained or latency-sensitive settings, a grid-feature-only model is a defensible and computationally efficient choice, requiring no image rendering or deep feature extraction at inference time.

4.7 Summary

In this chapter, we present an automated approach for algorithm selection based on supervised learning. We show that algorithm selection can be effectively predicted based on a preferred performance metric using environmental features in the pathfinding domain. Models trained on grid-based and CNN-extracted features achieved prediction accuracies ranging from 52% for path length optimization to 84% for memory consump-

tion minimization. Feature importance analysis revealed that start-goal distance, grid size, and obstacle density are strong predictors of algorithm performance, with start-goal distance emerging as the most influential factor across all performance criteria.

These results confirm that the performance variability observed in Chapter 3 follows learnable patterns: Our trained models successfully capture that environmental features can guide algorithm selection in place of expert knowledge or hand-crafted heuristics. Moreover, combining manually defined grid features with CNN-extracted features yields a more comprehensive representation of the environment than either feature type alone.

Despite the results obtained, the study approach exhibits several limitations: on one hand, our evaluation used fixed algorithm-specific parameters (lookahead parameter = 250 nodes for LRTA* and RTAA*; ARA* weight = 2.5); and changing these parameters might influence the performance of the algorithms and the trained models. On the other hand, obstacle configuration, including both their distribution and density, might affect the model's performance if changed; more broadly, although the MAPF benchmark already encompasses a wide variety of map types and scenarios, alternative training settings may still reveal different degrees of ambiguity, and investigating the sensitivity of the observed ties to training set composition remains a direction for future work. And lastly, our study in this chapter focused exclusively on static pathfinding scenarios. The trained models operate as one-shot predictors that select an algorithm given the initial environmental features, and cannot revise their choice once execution begins. Since the algorithm performance depends on the environment instance, any subsequent change in the environment, such as a moving obstacle, may result in a new problem instance based on Rice's framework [44], for which the original prediction is no longer guaranteed to remain optimal, even though it does not necessarily fail. The one-shot predictor developed here and the sequential meta-reasoner of the following chapters, therefore, solve structurally different problems and are not directly comparable as competing selectors.

This static, one-shot limitation motivates the architectural design introduced in Chapter 5: rather than predicting once before execution, algorithm selection is recast as a sequential decision-making process in which the agent continuously monitors environmental conditions and revises its algorithmic choices at runtime. This is not a limitation of learning as a tool but of the architectural framing: resolving it requires embedding selection inside the agent's deliberation cycle, which Chapter 5 undertakes.

Chapter 5

Adaptive BDI Architecture with Meta-Reasoning

5.1 Introduction

The preceding chapters established two complementary findings. chapter 3 demonstrated that no single algorithm consistently dominates across environmental conditions, confirming that algorithm performance is context-dependent. Chapter 4 showed that this context-dependence is learnable: supervised models trained on environmental features can predict optimal algorithm choices with reasonable accuracy for static problem instances. However, supervised models commit to a single algorithm before execution begins and thus cannot adapt if the environment changes, solutions become invalid, or the environment shifts in ways not anticipated at training time. Together, these findings define a precise architectural problem: we require an agent capable of continuously monitoring its environment and revising algorithmic choices at runtime to adapt as environmental conditions evolve. To address this constraint, we reconsider algorithm selection as an internal, adaptive component of the agent's decision-making cycle. This chapter investigates how adaptive algorithm selection can be realized within a deliberative agent architecture while preserving transparency of the reasoning layer and the interpretability of the conditions under which each selection is made, thereby addressing **RQ3**. The proposed approach treats algorithm selection as a meta-level decision within the agent's reasoning loop, grounded in structured belief-intention state, in contrast to the one-shot external prediction model of Chapter 4. A learning component operates over structured representations of the agent's cognitive state and dynamically selects among alternative algorithmic strategies at runtime. In this way, the choice

of algorithm becomes an explicit meta-decision grounded in the agent's deliberative context.

The chapter is organized as follows. We first establish the core capabilities and architectural design principles required for adaptive algorithm selection (§5.2). We then assess cognitive architectures and learning paradigms against these requirements, justifying the selection of BDI combined with reinforcement learning (§**??**). Finally, we present the proposed architecture, its state representation, and concrete example, showing how algorithm selection integrates into the BDI control loop (§5.4-**??**).

5.2 Requirements for Adaptive Algorithm Selection

Before committing to an architectural approach, we establish the core requirements that the proposed architecture must satisfy. These consist of five capabilities—what the system must accomplish—and five design principles—how it must accomplish them:

5.2.1 Core Capabilities for Adaptive Algorithm Selection

Runtime Algorithm Selection. Chapter 4 demonstrated that environmental features can predict optimal algorithms for static problem instances. However, in dynamic environments where conditions change during execution, pre-deployment predictions become invalid. Therefore, algorithm selection must be done at runtime, not at design time.

Context-Aware Selection. The feature importance analysis of Chapter 4 further showed which environmental characteristics are most informative depending on the optimization criterion. Consequently, runtime algorithm selection should be informed by both the agent's current environmental context and its optimization objective.

Sequential Decision-Making. Since conditions evolve during execution, selections made under one context (chapter 4) may become suboptimal (chapter 3) as new conditions emerge. Runtime algorithm selection (above) must therefore be treated as a sequential decision process where each selection is grounded in the current state and remains subject to revision as conditions evolve.

Learning from Experience While Chapter 4 demonstrated that algorithm selection can be learned from historical data for static problem instances, runtime adaptation requires the selection policy to continue improving as new situations are encountered during execution. The architecture should therefore learn from the outcomes of previous algorithm selections, progressively refining future decisions through experience.

Integration with Deliberation. Adaptive algorithm selection must be integrated into the agent's deliberation process rather than operating as an isolated optimisation module. Nevertheless, the deliberation process remains responsible for determining the agent's objectives and coordinating its reasoning, while meta-reasoning is responsible for selecting the computational procedure most appropriate for the current decision context—for example, choosing the planning algorithm used to solve the current planning problem. Maintaining this separation of responsibilities allows adaptive algorithm selection to be incorporated without changing the underlying deliberative model.

5.2.2 Architectural Design Principles

Deliberative Transparency The architecture must maintain inspectable representations of the agent's knowledge, goals, and reasoning state, making every action have a reasoning trace. The human operator must understand the agent's current beliefs, pursued goal, which algorithm is selected, and based on what, and why the conditions that favoured it no longer hold. This transparency is essential for trust, debugging, and verification in safety-critical applications, enabling operators to diagnose failures and validate correct behavior.

Interpretability of Selections Because each selection is grounded in a structured representation of the agent's beliefs and intentions, the *conditions* under which an algorithm was chosen are inspectable: one can recover the belief–intention state present when algorithm X was selected at time t. This contrasts with end-to-end learned policies (e.g., deep reinforcement learning for navigation), whose inputs are entangled in opaque sensory encodings. Inspecting these conditions does not, however, explain the learned mapping itself; why the policy favours one algorithm over another is not directly interpretable, and its post-hoc explanation, along with empirical validation through user studies or formal interpretability analysis, is left to future work (Section 7.2).

Modularity The meta-reasoning mechanism should be designed as a self-contained component that integrates into the BDI deliberation cycle without requiring redesign of its other components. This will enable us to develop and test the meta-reasoning independently before integration, evaluate different meta-reasoning approaches within the same deliberative framework, and to maintain it without requiring changes for other components.

Computational Tractability Meta-reasoning overhead must remain reasonable relative to algorithm execution cost. If the time required to select an algorithm approaches or exceeds the time required to run that algorithm, then the approach becomes im-

practical. However, this overhead can be acceptable if adaptive selection significantly improves performance. Therefore, the architecture should employ compact state representations that avoid redundant feature computation, fast selection mechanisms, and caching strategies when states change incrementally.

Generalization The architectural principles should be designed to apply to any domain in which multiple algorithms exist to solve the same task. This is a design target guiding the architecture presented below, not a claim validated in this thesis, whose empirical evaluation is confined to pathfinding (Chapter 6). Representative, untested examples of domains the design is intended to accommodate include:

- Scheduling: selecting among different scheduling heuristics (earliest deadline first, shortest job first, etc.) based on current workload characteristics
- Resource allocation: choosing among load balancing strategies (round-robin, least-connections, weighted) based on server loads and traffic patterns
- Constraint solving: selecting among SAT solvers or CSP algorithms based on problem structure
- LLM-based agents: choosing among retrieval strategies, reasoning modes, or tool invocation patterns based on query complexity and available context

5.3 Architecture Selection

Having established the requirements, we now assess alternative cognitive architectures and learning paradigms to justify the selection of BDI integrated with reinforcement learning as the most suitable approach.

5.3.1 Cognitive Architectures

Reactive architectures like subsumption [11] prioritize fast, layered behaviors without explicit beliefs or goals, leading to opaque decision-making that fails our deliberative transparency requirement. Layered architectures such as 3T [6] blend reactive and deliberative layers for flexibility but introduce integration complexity across layers, hindering modularity. BDI [42] explicitly models beliefs (world knowledge), desires (goals), and intentions (committed plans) in a cyclic deliberation process. This provides explicit representations for transparency and interpretability, modular deliberation phases, and natural goal-directed reasoning, though it lacks native learning mechanisms. Recent

LLM-based architectures offer powerful reasoning but encode decisions through distributed parameters rather than formal cognitive states [57]. BDI, by construction, exposes beliefs and intentions as inspectable structures, directly satisfying our transparency and interpretability requirements. LLM-based agents remain a promising direction for future work, as discussed in Section 7.2. SOAR and ACT-R [31, 3] offer learning through unified production rules or memory models, yet their full cognitive simulation adds unneeded complexity for targeted goal-directed tasks.

5.3.2 Learning Paradigms

Beyond the choice of cognitive architecture, the integration of a learning mechanism is necessary to satisfy the learning from experience and sequential decision-making requirements. We assess four paradigms against these needs.

Supervised learning relies on static labeled data and predicts single outputs without considering action sequences or long-term outcomes. As demonstrated in chapter 4, supervised learning achieved 78-84% accuracy for predicting optimal algorithms on static pathfinding instances; however, this paradigm exhibits three fundamental limitations for dynamic environments. First, supervised models commit to a single selection before execution begins (*one-shot prediction*) and cannot revise this choice as conditions evolve. Second, they offer no mechanism for temporal credit assignment: a model cannot learn that an algorithm chosen at time t caused a failure at time $t + k$. Third, they assume static feature distributions, whereas in dynamic pathfinding, obstacle density and path validity change continuously as obstacles move and plans become invalid.

Collectively, these constraints show that supervised learning treats algorithm selection as a classification problem over static feature vectors, whereas dynamic environments require sequential decision-making over evolving state trajectories.

Evolutionary algorithms optimize via population-based search but demand vast samples for convergence, making them impractical for the sample-efficient, sequential setting required here. Imitation learning requires expert trajectories, which are often unavailable or costly to obtain, and struggles with distribution shift when encountering unseen states, offering no mechanism for autonomous learning from experience. Reinforcement learning naturally models sequential decision problems as Markov Decision Processes, maximizing cumulative reward through policies learned via trial-and-error. Temporal-difference methods handle delayed rewards efficiently and support sample-efficient exploration.

5.3.3 Selected Approach

BDI provides the deliberative structure, transparency, and goal-directed reasoning that our requirements demand: its explicit representation of beliefs, desires, and intentions grounds algorithm selection in inspectable cognitive state. However, BDI alone offers no mechanism for learning from experience or adapting selection strategies over time. Reinforcement learning complements BDI by enabling the system to improve its policy through environmental interaction. Standalone RL, however, provides no deliberative structure, no goal representation, and no symbolic transparency. The two are complementary by design.

Table 5.1 confirms that neither component alone satisfies all requirements, whereas their integration does.

Table 5.1: Satisfaction of requirements by the proposed architecture

Requirement	BDI Alone	RL Alone	BDI + RL
Core Capabilities			
Runtime algorithm selection	✗	✓	✓
Context-aware selection	✓	✗	✓
Sequential decision-making	✗	✓	✓
Learning from experience	✗	✓	✓
Integration with deliberation	✓	✗	✓
Architectural Principles			
Deliberative transparency	✓	✗	✓
Interpretability of selections	✓	✗	✓
Modularity	✓	✓	✓
Computational tractability	✓	✓	✓
Generalization	✓	✓	✓

5.4 Architecture Overview

In this section, we present an extended BDI architecture that integrates a meta-reasoning layer into the agent's deliberation cycle, enabling runtime algorithm selection based on the agent's current belief-intention state.

Building on the BDI cycle introduced in Section 2.4, classical BDI architectures typically perform means–end reasoning using either a fixed planning algorithm or a predefined symbolic plan library, limiting their ability to adapt when the environment changes in ways unforeseen by the developer.

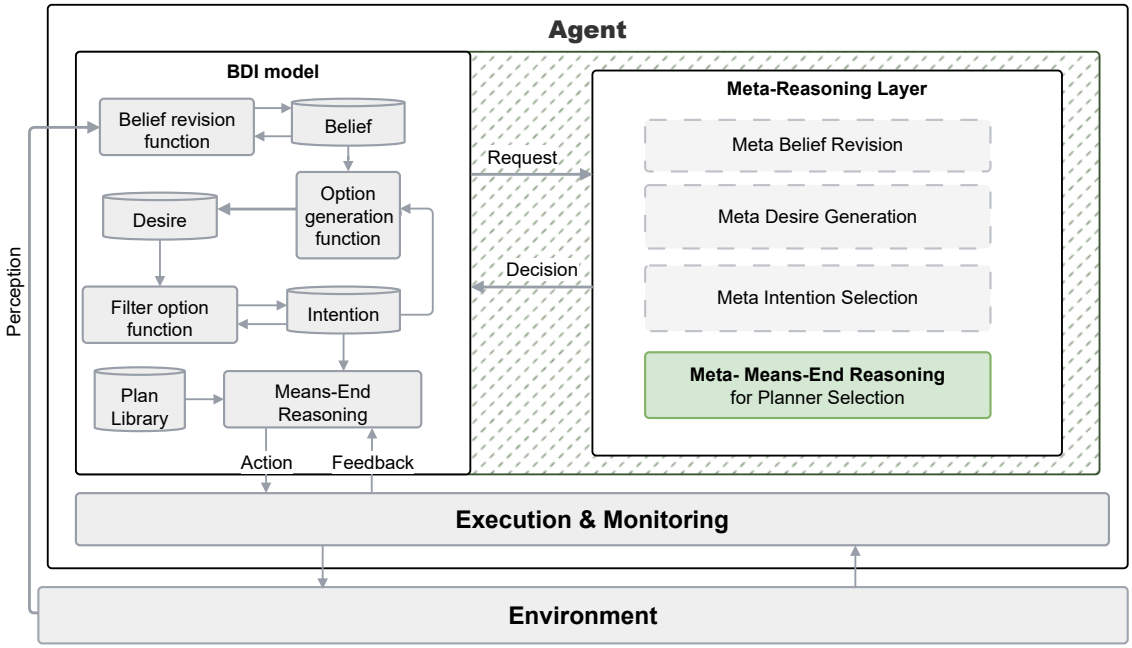


Figure 5.1: Meta-reasoning enhanced BDI architecture (Deployment Phase). Left: BDI core extended with meta-reasoning interface. Right: meta-reasoning layer showing modules for each BDI component.

To address this limitation, we introduce a meta-reasoning layer that integrates a reinforcement learning policy into the BDI reasoning loop. At runtime, the meta-reasoner receives a representation of the current belief-intention state and selects the algorithm predicted to yield the best long-term outcome, based on a policy learned through prior interaction with the environment. This provides an adaptive, experience-driven selection mechanism while preserving the symbolic transparency of the BDI deliberation cycle.

Figure 5.1 gives a structural overview of the proposed architecture. The left panel shows the traditional BDI core, where the symbolic reasoning functions operate independently of the meta-reasoning layer. The right panel introduces the meta-reasoning layer itself, which may support algorithm selection for multiple components of the BDI cycle.

Algorithm 1 illustrates the integration of Meta Means-End Reasoning, responsible for the selection of the planner, within the BDI loop. The algorithm itself is unchanged from the version provided and retains the full operational semantics of the classical BDI cycle [8]. The key extension appears when the agent calls `MetaMeansEndReasoning`, which selects the planning algorithm based on the current belief-intention state.

Algorithm 1 BDI Agent Deployment with Meta-Reasoning

1: **Input:** B_0, I_0, $\mathcal{A} = \{a_1, \ldots, a_k\}$, trained meta-reasoner $\mathcal{M}$
2: {Instantiate BDI Agent}
3: $B \leftarrow B_0$, $I \leftarrow I_0$, $Plan \leftarrow null$
4: **while** True **do**
5: $B \leftarrow brf(B, percepts())$
6: $D \leftarrow options(B, I)$
7: $I \leftarrow filter(B, D, I)$
8: $a^* \leftarrow MetaMeansEndReasoning(B, I, \mathcal{A}, \mathcal{M})$
9: $Plan \leftarrow plan(B, I, a^*)$
10: **while** not (empty $(Plan)$ **or** succeeded(I, B) **or** impossible(I, B)) **do**
11: $\alpha \leftarrow$ first$(Plan)$
12: execute(α)
13: $Plan \leftarrow$ tail$(Plan)$
14: $B \leftarrow brf(B, percepts())$
15: **if** reconsider(I, B) **then**
16: $D \leftarrow options(B, I)$
17: $I \leftarrow filter(B, D, I)$
18: **end if**
19: **if** not sound$(Plan, I, B)$ **then**
20: $a^* \leftarrow MetaMeansEndReasoning(B, I, \mathcal{A}, \mathcal{M})$
21: $Plan \leftarrow plan(B, I, a^*)$
22: **end if**
23: **end while**
24: **if** succeeded(I, B) **then**
25: **return** success
26: **end if**
27: **end while**

Extended description of Algorithm 1. The algorithm preserves the standard BDI control loop while adding a meta-level decision point. The first part of the loop (belief update, option generation, intention filtering) operates exactly as in classical BDI systems. The extension appears at line 8, where the agent queries the meta-reasoner to select the most suitable planning algorithm for the current belief–intention configuration, based on a learned policy that estimates each planner's expected performance in the current state. We distinguish two levels of action. The meta-level action $a \in \mathcal{A}$ is the choice of planning algorithm returned by the meta-reasoner; the object-level actions α are the primitive moves produced by the selected planner and executed in the environment. The meta-reasoner operates only at the former level, leaving plan execution unchanged.

5.4.1 State Representation: Extending Rice's Framework

As introduced in Section 2.1, Rice's algorithm selection framework [44] formalizes selecting the most appropriate algorithm for a problem instance based on descriptive features. However, Rice's formulation assumes static problem instances with fixed feature sets, making it unsuitable for dynamic environments where problem states evolve during execution.

We extend Rice's framework to sequential decision-making by embedding algorithm selection within the agent's cognitive state. Rather than extracting static features once, we represent the agent's state at each time step t as the pair (B_t, I_t), where B_t denotes the agent's belief base and I_t denotes its currently active intention at step t. The meta-reasoning state is then defined as:

$$s_t = \langle f_{\text{env}}(B_t),\ f_{\text{goal}}(I_t),\ f_{\text{context}}(t) \rangle,$$

where **environmental features** f_{env} capture observable characteristics of the environment from the agent's beliefs (corresponding to Rice's instance features); **goal-related features** f_{goal} encode the deliberative context derived from the agent's current intentions; and **execution context** f_{context} captures temporal dependencies such as previously selected algorithms, partial plan validity, and recent environmental changes, a component absent from static frameworks but critical in dynamic settings where past decisions influence future performance.

This formulation generalizes Rice's per-instance mapping to a sequential mapping over evolving deliberative states. In particular, the inclusion of execution context enables the meta-reasoner to capture path dependency effects: for example, a deep-lookahead planner may be appropriate in sparse environments, but suboptimal if a previously generated plan has already become partially invalid. Without context-aware features, such distinctions would be lost.

By grounding the state representation in beliefs and intentions, algorithm selection becomes an explicit component of the agent's reasoning process rather than an external predictive module. The meta-reasoner is additive: it determines only which algorithm to invoke, leaving belief revision, intention commitment, and plan execution unchanged. The agent's deliberative structure therefore remains symbolic and inspectable. Unlike approaches that replace symbolic plan generation with a learned policy [57, 37, 12], here the learned component is confined to the selection decision, leaving the BDI deliberation cycle intact. This is how the architecture addresses RQ3: every selection is conditioned on the explicit, inspectable belief-intention state, so the full context of each decision is

transparent to a human operator. The learned mapping from that state to an algorithm is not itself interpretable; this distinction, and approaches to narrowing it, are discussed in Chapter 7.

The concrete encoding of these feature components for the pathfinding domain is detailed in Chapter 6, where we describe how spatial, goal-oriented, and temporal signals are combined into a vector representation suitable for Deep Q-Network training.

5.4.2 Illustrative Example

To illustrate how the architecture operates at runtime, consider an agent with intention I to reach a goal in a dynamic environment.

1. **Initial state.** The agent's beliefs B_0 encode a low-complexity environment. `MetaMeansEndReasoning` (line 8) constructs $s_1 = \langle f_{\text{env}}(B_0), f_{\text{goal}}(I), f_{\text{context}}(1) \rangle$ and selects algorithm a_1^*, suited to current conditions. The selected planner a_1^* produces a plan $Plan$ (a sequence of actions), and execution begins.

2. **Environmental change.** Conditions change mid-execution, invalidating $Plan$, in a path planning scenario this could be due presenting new moving obstacles in the environment. The belief revision function updates B, and `not sound`$(Plan, I, B)$ becomes true (line 21).

3. **Adaptation.** `MetaMeansEndReasoning` (line 22) re-evaluates the portfolio under the updated belief-intention state and selects a more suitable algorithm a_2^* for current environment characteristics. Thus, a new plan $Plan'$ is generated and execution resumes from where it stopped.

Throughout all three steps, intention I remains unchanged. The meta-reasoning layer adapts *how* the goal is pursued without altering *what* is pursued, preserving the goal-directed structure of the BDI deliberation cycle.

5.4.3 Reward Function Design Principles

The reward function plays a central role in enabling adaptive meta-level control. Unlike standard reinforcement learning settings, where rewards evaluate the consequences of primitive environment actions, our reward signal evaluates algorithmic decisions according to properties such as solution quality, computational cost, and goal-directed

progress, where applicable. This makes the reward signal domain-sensitive: which objectives are relevant, and how they are measured, depends on the algorithm portfolio under consideration.

At the architectural level, we consider meta-level evaluation as a multi-objective problem. Each algorithmic decision is associated with a vector-valued reward

$$\mathbf{r}_t = \big(r_t^{(1)},\ r_t^{(2)},\ \ldots,\ r_t^{(K)}\big),$$

whose $K \geq 1$ components $r_t^{(k)}$ capture objectives relevant to the algorithm-selection task at hand (for example, solution quality, computational cost, or goal-directed progress). To remain compatible with standard single-objective learners such as DQN, this vector is reduced to a scalar training signal through linear scalarization:

$$r_t = \sum_{k=1}^{K} w_k\, r_t^{(k)},$$

where $\mathbf{w} = \big[\, w_1, \ldots, w_K \,\big]^\top$ are domain-specific weights balancing the objectives. The number of objectives K and their specific formulation depend on the portfolio and domain, not on the architecture. We use linear weighted-sum scalarization because it is the most common and interpretable method for converting a reward vector into a single training signal, with each weight directly reflecting the importance of its objective. Other scalarizations, such as lexicographic or Pareto-based methods [58], could be used without altering the underlying objectives; we leave their comparison to future work.

This formulation explicitly encodes the inherent tension in algorithm selection: high-quality solutions often require greater computation, while rapid progress may sacrifice optimality. By adjusting the weight configuration, the meta-policy can be biased toward optimality, reactivity, or computational efficiency depending on deployment requirements; for instance, real-time systems may prioritise efficiency, whereas safety-critical domains may emphasise solution quality. The concrete weight values used in our pathfinding instantiation, together with the rationale for their relative magnitudes, are reported in Appendix B.1.

Designing a stable and meaningful meta-level reward presents several challenges. Poorly calibrated weights may induce oscillatory behaviour, in which the agent switches between algorithms without converging to a stable strategy; conflicting components may systematically favour one class of algorithms over another regardless of context; and overemphasis on immediate progress may discourage exploration of computationally expensive but globally beneficial strategies. The reward structure must therefore be

balanced so that the learned policy captures long-term utility rather than short-term gain.

In our pathfinding instantiation (chapter 6), this abstract structure is concretised with $K = 6$ components, measured by quantities such as number of expanded nodes, blocked moves, and plan utilisation (Section 6.4). The weighted-sum structure serves as a transferable design template, but the applicability of each term must be assessed per domain. For instance, in tool selection for LLM-based agents, goal-directed progress may have no natural operationalisation and should be omitted or replaced with a domain-appropriate measure such as task completion confidence.

By framing algorithm selection as a reinforcement learning problem with a structured multi-objective reward, we enable the BDI agent to learn a meta-policy that adapts its choice of algorithm to the current problem context, combining experience-driven adaptation with the deliberative structure established in Section 5.4.

5.5 Summary

In this chapter, we have presented the architectural contribution of the thesis, addressing **RQ3**. We established ten requirements motivating the integration of BDI with reinforcement learning, and showed that this combination satisfies requirements that neither component alone can meet. The proposed architecture incorporates adaptive algorithm selection through a learned meta-reasoner invoked by the BDI deliberation cycle, with each selection based on the agent's current belief-intention state while leaving the deliberative structure intact.

The proposed architecture [22] extends the classical BDI cycle with a meta-reasoning layer that can be invoked at any stage of the BDI deliberation cycle. Algorithm selection is grounded in a structured state representation derived from the agent's current belief-intention, forming the state feature representation in Rice's framework. The reward structure operates at the meta-level, evaluating algorithmic choices rather than primitive actions.

Chapter 6 instantiates this architecture concretely in a dynamic pathfinding domain [22], implementing the state encoding, reward function, and DQN training procedure, and evaluating the resulting meta-policy against fixed-algorithm baselines.

Chapter 6

Experimental Evaluation of RL-Based Meta-Reasoning

6.1 Introduction

Chapter 5 proposed an extended BDI architecture in which a meta-reasoning layer selects among alternative planning algorithms at runtime, grounding selection in the agent's current belief-intention state. This chapter instantiates that architecture concretely in a dynamic grid-based pathfinding domain. The meta-reasoner is instantiated using a Deep Q-Network (DQN), trained through episodic interaction with a dynamic grid-based pathfinding environment to learn a policy that maps belief-intention states to planner choices. The algorithm portfolio consists of four variants of Real-Time Adaptive A* (RTAA*) with lookahead depths of 4, 8, 32, and 128, representing a range of trade-offs between planning depth and computational cost. Unlike the static, one-shot predictors evaluated in Chapter 4, the learned policy can revise algorithmic choices across the course of an episode as environmental conditions evolve. Both this chapter and Chapter 5 extend our prior work [22].

The chapter proceeds as follows. Section 6.2 formalizes algorithm selection as a sequential decision-making problem. Section 6.3 describes the DQN instantiation and its integration into the BDI means-end reasoning phase. Section 6.4 presents the experimental setup, results, and analysis. Section 6.5 discusses findings and limitations. Section 6.6 summarizes the chapter.

6.2 Sequential Formulation of Algorithm Selection

As discussed in Chapter 5, Rice's classical algorithm selection framework [44] captures the static per-instance case, whereas pathfinding in dynamic environments evolves over time and requires sequential decision-making. This section formalizes that sequential setting.

We extend the classical formulation to dynamic settings. We denote by p_j a problem instance drawn from a distribution ρ, and let $p_j(t)$ denote its state at time $t \in \{0, 1, \ldots, T_j\}$, where t indexes the successive meta-level decision points at which an algorithm is selected (in our instantiation, the planner invocations), rather than the primitive execution steps of the underlying algorithm. The episode terminates at T_j either because the problem instance is successfully completed (e.g., the goal is reached in pathfinding) or because a failure condition is met (e.g., a step or computational budget is exceeded); both outcomes are handled uniformly within the same framework, since the reward function can assign positive terminal rewards to success and negative ones to failure, as instantiated in Section 6.3. We write $\mathcal{P}$ for the space of all problem states, so that $p_j(t) \in \mathcal{P}$. Let $\mathcal{A} = \{a_1, \ldots, a_k\}$ be a portfolio of k algorithms, where $i \in \{1, \ldots, k\}$ indexes individual algorithms. The reward obtained by applying algorithm a_i to problem state $p_j(t)$ is:

$$r_t = \mathcal{R}(a_{i_t}, p_j(t)) \in \mathbb{R},$$

where $\mathcal{R} : \mathcal{A} \times \mathcal{P} \to \mathbb{R}$ is the reward function (i.e., scalar performance metric) and $i_t \in \{1, \ldots, k\}$ denotes the index of the algorithm selected at step t.

Rather than committing to a fixed algorithm, we apply a sequence of algorithm choices $\mathcal{S} = (a_{i_0}, a_{i_1}, \ldots, a_{i_{T_j}})$, where each a_{i_t} is drawn from $\mathcal{A}$; the cumulative return of $\mathcal{S}$ on problem instance p_j is:

$$J(\mathcal{S}, p_j) = \sum_{t=0}^{T_j} r_t = \sum_{t=0}^{T_j} \mathcal{R}(a_{i_t}, p_j(t)).$$

To enable adaptive selection, we define a policy $\pi : \mathcal{P} \to \mathcal{A}$ that maps problem states to algorithm choices, and let Π denote the space of all such (deterministic) policies; restricting attention to deterministic policies is standard for value-based methods, whose learned policy is the greedy policy with respect to the estimated action values [54]. The cumulative performance of a policy π on p_j is:

$$J(\pi,\, p_j) = \sum_{t=0}^{T_j} \mathcal{R}(\pi(p_j(t)),\, p_j(t)).$$

The objective is to learn an optimal policy π^* that maximises expected cumulative return across problem instances drawn from ρ:

$$\pi^* = \arg\max_{\pi \in \Pi} \mathbb{E}_{p_j \sim \rho}[J(\pi, p_j)]\,.$$

where $\mathbb{E}_{p_j \sim \rho}[\cdot]$ denotes the expectation over problem instances p_j sampled from the distribution ρ.

This sequential formulation naturally lends itself to reinforcement learning: algorithm selection becomes an action, the problem state $p_j(t)$ defines the RL state, and r_t is the reward signal. The transition dynamics — how the next state $p_j(t{+}1)$ arises from the current state, the execution of the selected algorithm, and exogenous environmental change — are not modelled explicitly: the formulation is model-free, and the learning procedure of Section 6.3 estimates action values directly from sampled transitions, without requiring an explicit transition function. Summation of per-step rewards is the standard RL return, which correctly aggregates the reward sequence regardless of the specific metric encoded by $\mathcal{R}$. The concrete instantiation of $\mathcal{R}$ for the pathfinding domain is given in Section 6.3. Crucially, this formulation imposes no domain-specific assumptions. The problem state $p_j(t)$ is an abstract representation that can instantiate any environment grid cells, resource queues, or task graphs, provided it captures features relevant to distinguishing algorithm performance. The action space $\mathcal{A}$ is the algorithm portfolio, which may contain pathfinders, schedulers, or any set of candidate solvers. The reward r_t can encode arbitrary performance metrics such as solution quality or others. While Section 6.3 instantiates these quantities concretely for pathfinding, the sequential decision framework itself is fully domain-agnostic: it casts algorithm selection as a finite-horizon Markov Decision Process regardless of the underlying application. Because every episode terminates at T_j, the cumulative return J is well defined without discounting; the DQN implementation nonetheless employs a discount factor of $\gamma = 0.99$ (Table 6.1), a standard choice that stabilises value estimation while closely approximating the undiscounted finite-horizon objective.

The learned policy is integrated into the BDI means–end reasoning phase (Algorithm 1), as described in Chapter 5, where the meta-reasoner selects the planning algorithm based on the agent's current beliefs and intentions.

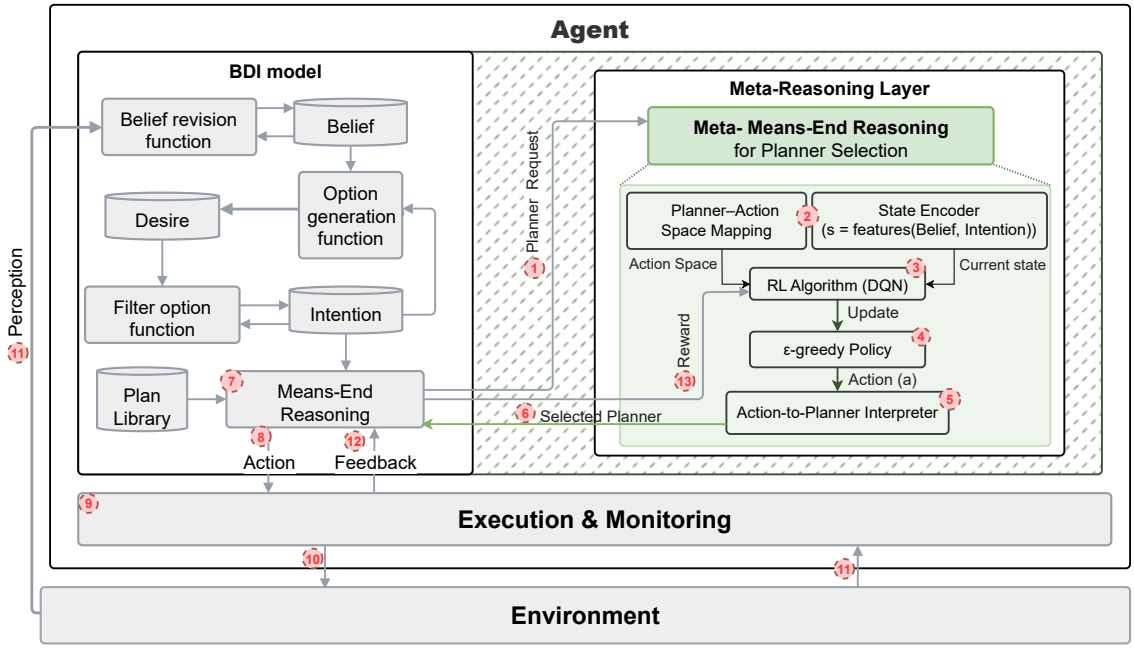


Figure 6.1: Training workflow for the meta-reasoner (Training Phase). Left: execution flow through the BDI components. Right: RL training loop with feature encoding, algorithm selection, reward evaluation, and policy updates.

6.3 Integrating Deep Q-Network in Meta Means–End Reasoning

In the following, we instantiate the meta-reasoner using reinforcement learning with Deep Q-Network (DQN) algorithm. Each planning algorithm in $\mathcal{A}$ is mapped to a discrete RL action, allowing the meta-reasoner to treat algorithm selection as an action-selection problem. Given the encoded belief–intention state s, the Deep Q-Network evaluates the expected long-term utility of selecting each planner. The planner chosen by the agent corresponds to the action that maximises the Q-value:

$$a^* = \arg\max_a Q_\theta(s, a),$$

where Q_θ is the trained DQN. The planner associated with a^* is then passed to the `plan` function to generate the course of action executed within the BDI cycle.

The meta-reasoner learns from experience which planning algorithm is most suitable for different belief–intention states by observing the outcomes of past algorithm selections across diverse scenarios.

Figure 6.1 depicts the learning workflow used to train the meta-reasoner. The left

Algorithm 2 Train Meta-Reasoner for Path Planning

Input: $\mathcal{A} = \{a_1, \ldots, a_k\}$, M {algorithm portfolio, #training episodes}
Output: Trained Q_θ {for path planning intentions}
Initialize Q-network parameters θ, replay buffer $\mathcal{D}$, and ϵ for exploration
for episode $e = 1$ to M **do**
  $B \leftarrow B_0$, $I \leftarrow I_0$ {reset environment with path planning task}
  $s \leftarrow \text{features}(B, I)$
  $Done \leftarrow False$
  while not $Done$ **do**
    Select $a \in \mathcal{A}$ using ϵ-greedy over $Q_\theta(s, a)$
    $Plan \leftarrow plan(B, I, a)$
    Execute the plan and observe the outcome
    $B \leftarrow brf(B, percepts())$
    $I \leftarrow filter(B, options(B, I), I)$
    $s' \leftarrow \text{features}(B, I)$
    $r \leftarrow computeReward(outcome, B, I)$
    Store (s, a, r, s') in $\mathcal{D}$
    Sample minibatch from $\mathcal{D}$
    Update θ using DQN loss
    $s \leftarrow s'$
    if empty($Plan$) or succeeded(I, B) or impossible(I, B) **then**
      $Done \leftarrow True$
    end if
  end while
end for
return θ

panel highlights how the BDI cycle unfolds during training, while the right panel illustrates the reinforcement learning loop. The numbered arrows indicate the flow of states, actions, rewards, and network updates as described in Algorithm 2. The figure emphasises the closed-loop interaction between symbolic reasoning, planning algorithms, environmental feedback, and the RL-based meta-reasoner.

The remainder of the loop executes the plan, checking after each action whether the plan has become unsound or whether intentions should be reconsidered. If the agent encounters unexpected changes in the environment (e.g., obstacles, blocked paths, inconsistent beliefs), it does not blindly replan using the same algorithm; instead, it allows the meta-reasoner to select a new algorithm, making planning itself an adaptive, context-sensitive capability.

Before the meta-reasoner can be used at runtime, it must be trained. Algorithm 2 describes the training procedure used for the meta-reasoner. The algorithm defines how

experience is collected through episodes in which the agent repeatedly selects planners, executes resulting plans, observes outcomes, and updates the Q-network. The reward function measures both planning efficiency and execution success, guiding the network towards policies that balance solution quality, computational cost, and robustness across varied scenarios.

Extended description of Algorithm 2. The training procedure iteratively refines the Q-function through episodic interaction with the environment. At decision step, the meta-reasoner selects a random planner with probability ε and otherwise chooses the planner with the highest estimated Q-value, i.e., $\arg\max_a Q_\theta(s, a)$. This ε-greedy strategy was adopted to balance exploration of different planning algorithms with exploitation of the currently learned policy [35]. Exploration is particularly important in this setting because the relative performance of planners varies across environmental conditions, and restricting training to greedy selection could prematurely bias the policy toward suboptimal planner choices in underexplored conditions. The exploration rate ε decays linearly from 1.0 to 0.05 over the first 33% of training steps, ensuring broad initial exploration before converging to an exploitative policy.

Experience tuples (s, a, r, s') are stored in a replay buffer $\mathcal{D}$, and mini-batches of 128 transitions are sampled uniformly at random from $\mathcal{D}$ at each update step [35]. Uniform sampling decorrelates consecutive experiences and stabilises gradient updates. Over many episodes, the learned Q-function converges towards a policy that identifies the planning algorithm most likely to yield strong performance for any given belief-intention state.

6.3.1 Implementation and Hyperparameters

The DQN meta-reasoner was implemented using Stable-Baselines3 [41] with a `MultiInputPolicy` to accommodate the mixed observation space combining a stacked spatial tensor with scalar contextual features. The Q-network architecture and full training configuration are reported in Table 6.1. The exploration fraction of 0.33 was chosen deliberately to sustain planner diversity throughout training, given the non-stationary nature of the environment in which no single algorithm is consistently optimal. Training convergence was verified by monitoring rollout reward, evaluation reward, TD loss, and exploration rate; the resulting diagnostics are reported in Appendix B.2.

Table 6.1: DQN hyperparameters used for training the meta-reasoner.

Hyperparameter	**Value**
Algorithm	DQN (Stable-Baselines3)
Policy	`MultiInputPolicy`
Learning rate	1×10^{-4}
Discount factor (γ)	0.99
Replay buffer size	200,000 transitions
Batch size	128
Learning starts	5,000 steps
Train frequency	every 4 environment steps
Gradient steps per update	2
Target network update interval	1,000 steps
Initial exploration ε	1.0
Final exploration ε	0.05
Exploration fraction	0.33 (of total training steps)
Total training steps	5×10^6
Q-network hidden layers	$[256, 256]$
CNN layers	Conv(32, 3×3) $\rightarrow$ Conv(64, 3×3)
Features dimension	256
Random seed	42

6.4 Experimental Validation

6.4.1 Setup

This section describes the environment used for our experiments, the baseline planning algorithms, the configuration adopted for training the meta-reasoner, and the metrics employed to evaluate the proposed architecture.

All experiments were conducted in a two-dimensional grid-based navigation domain of size 10×40, in which an agent must travel from a start location to a goal while avoiding both static and dynamic obstacles. In BDI terms, the agent's active intention is its commitment to reach the designated goal cell, which is why the goal position later appears explicitly in the meta-reasoner's observation vector. At the beginning of each episode, a set of moving obstacles is initialised with random positions, velocities, and accelerations. Their density varies between roughly 0.2% and 32% of the grid cells. The grid dimensions and obstacle density range were chosen to reflect typical corridor-navigation scenarios encountered in robotic applications, while providing sufficient variability to stress-test planners across sparse and highly congested configurations. As the episode unfolds, obstacles move within the grid and bounce off its boundaries, altering

the environment in ways that may invalidate previously viable paths.
All experiments were executed on an Intel i9 CPU (27 cores, 3.30 GHz) with 250 GB RAM under Ubuntu 22.04. The agent moves in the four cardinal directions; all planners use Manhattan distance as the heuristic, which is admissible under this movement model. The empirical evaluation is implemented on top of Gymnasium [55], an open-source toolkit that provides a common interface for defining reinforcement-learning environments and running experiments, and which serves as the environment layer for the runs reported in this chapter.

The meta-reasoner selects among four variants of RTAA*: RTAA*(4), RTAA*(8), RTAA*(32), and RTAA*(128), which differ solely in lookahead depth. Shallow lookaheads enable faster replanning but often lead to suboptimal routes, whereas deeper lookaheads typically produce better paths at higher computational cost. The agent must therefore learn to identify which lookahead setting is most appropriate given the current environmental state and the current state of the plan executed so far.

Selecting variants of a single algorithm family, rather than the broader set of six search algorithms evaluated in Chapter 4, is a deliberate design choice. It isolates planning depth as the sole axis of variation and removes the confounding effects that arise when comparing heterogeneous algorithm classes. This allows the meta-reasoner to learn whether different lookahead depths are appropriate for different environmental conditions, while ensuring that observed performance differences can be attributed to planner depth rather than to fundamental differences between unrelated search algorithms.

At each decision point, the meta-reasoner receives an observation vector that encodes both the spatial structure of the grid and temporal patterns in obstacle motion, as shown in Figure 6.2. The *spatial* component is a stacked tensor of shape $(4 \times 10 \times 40)$, consisting of three consecutive grid frames and one path-mask frame (1,600 values total). The *contextual* component comprises 14 scalar and categorical features: one-hot encodings of the current and previous planner (4+4 values), agent and goal positions (2+2 values), obstacle density (1 value), and a path validity score (1 value). The full observation vector has a total dimensionality of 1,614.

Since the spatial input stacks several full-grid frames, its dimensionality scales with the grid size, so adopting larger grids as those in Chapter 4 would have made training computationally expensive. We therefore opt for a smaller grid; the rectangular shape is deliberate: it keeps the start and goal far apart, making the navigation task more challenging for the planner.

The training procedure, described in Algorithm 2, uses a Deep Q-Network (DQN)

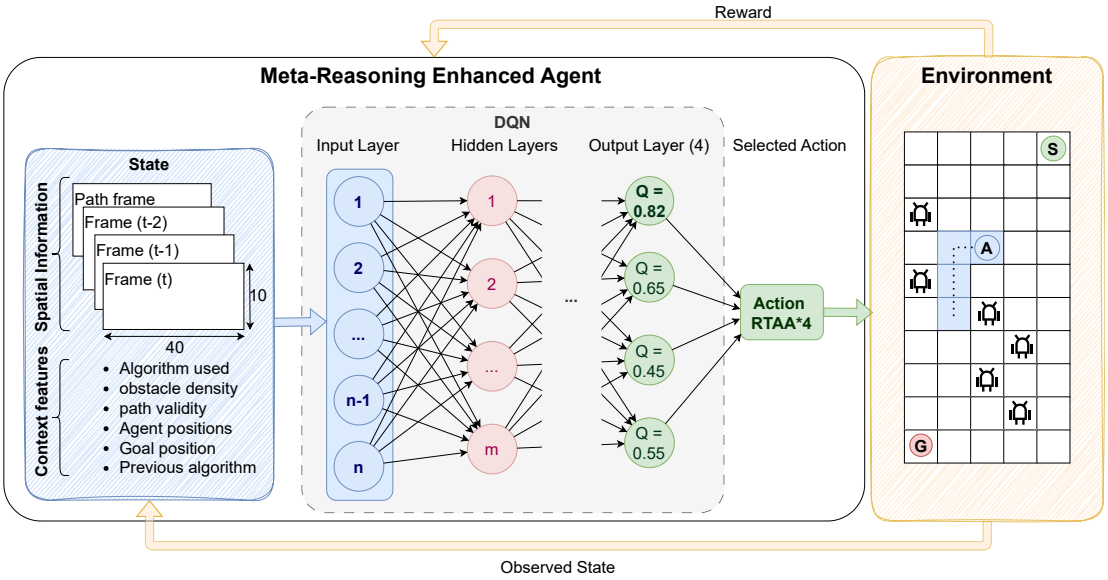


Figure 6.2: Deep Q-Network adopted inside BDI planner selection module. The DQN processes spatial and contextual features to select among RTAA* variants with different lookahead depths. Right: example navigation scenario in a 2D grid with dynamic obstacles and best selected action is RTAA*(4) with Q-value equals to 0.82.

to learn a policy for planner selection. At each step of an episode, the meta-reasoner selects a planner, which generates a partial path. The agent executes this path until it terminates or becomes invalid, after which beliefs are updated and a new state is extracted. A reward signal evaluates both planning and execution quality, ensuring that the learned policy reflects not only immediate progress but also overall path robustness and computational efficiency.

The abstract reward function $\mathcal{R}(a_{i_t}, p_j(t))$ introduced in Section 6.2 is instantiated here as a weighted combination of six components that reflect the specific performance trade-offs of the pathfinding domain:

$$r_t = r_{\text{planning\_cost}} + r_{\text{plan\_fail}} + r_{\text{nodes}} + r_{\text{efficiency}} + r_{\text{block}} + r_{\text{util}}.$$

These terms capture complementary aspects of performance. Planning invocations, search effort, and execution failures are penalised through $r_{\text{planning\_cost}}$, r_{nodes}, and $r_{\text{plan\_fail}}$, while r_{block} discourages blocked movements during execution. In contrast, $r_{\text{efficiency}}$ and r_{util} provide positive feedback for efficient use of lookahead and effective utilisation of generated plans.

The relative magnitudes of these components establish a clear hierarchy of objec-

tives. Terminal rewards (± 50) dominate cumulative per-step costs, ensuring that task completion remains the primary objective. At the planning level, a fixed penalty of -5 per invocation limits excessive replanning, while a proportional penalty of -0.01 per expanded node discourages wasteful search. A blocked-move penalty of -2 further penalises paths that fail during execution. Overall, this reward structure encourages the agent to balance solution quality, computational efficiency, and robustness to environmental changes. The numerical values of all reward coefficients are reported in Appendix B.1.

6.4.2 Results and Analysis

We evaluate the RL-based meta-reasoner against four fixed-planner baselines corresponding to the algorithms available in the model's action space: RTAA*(4), RTAA*(8), RTAA*(32), and RTAA*(128). These baselines represent the standard practice in a BDI system, where the agent commits to a single planning algorithm throughout execution. After training, we assessed all methods over 1,000 unseen test episodes. The latter were generated from the same parameter ranges as the training, but instantiated with new random seeds, ensuring that performance is assessed on conditions not encountered during training.

To obtain more statistically robust performance estimates, we then conducted an extended evaluation over 10,000 episodes to provide more reliable estimates of average performance across diverse scenarios. The extended evaluation covered scenarios combining both varying obstacle densities and different dynamic obstacle behaviour (i.e., obstacles move in different directions at different speeds). Table 6.2 reports the success rates, mean rewards, and median computational overhead (ms); for the trained agent, this is end-to-end decision latency including observation encoding and DQN inference, whereas for the fixed planners it reflects per-step planning time only. The generally high success rates across all methods (approximately 89–99%) indicate that most episodes are solvable even with fixed strategies using limited lookahead. This provides a meaningful test bed for assessing algorithmic trade-offs: although success rates are similar, the average rewards differ substantially, revealing deeper behavioural differences.

Although all rewards are negative due to inherent planning costs, the trained agent demonstrates substantially superior efficiency, outperforming all fixed baselines, and achieving both the highest success rate (99.43%) and the lowest negative average reward (-14.50). RTAA*(128) achieves comparable success (99.26%) but incurs a fourfold worse reward (-61.94). Its per-decision latency appears lower than the meta-reasoner

Table 6.2: Overall performance across 10,000 evaluation episodes. Success rate is the percentage of episodes reaching the goal; average reward is computed over all episodes; computational overhead is reported as median decision latency (for the trained agent: end-to-end meta-reasoning time; for fixed planners: per-step planning time).

Method	**Successful Episodes**	**Avg Reward (All)**	**Computational Overhead (ms)**
Trained agent	**9,943 (99.43%)**	−14.50	2.34
RTAA*(128)	9,926 (99.26%)	−61.94	1.46
RTAA*(32)	9,900 (99.00%)	−28.74	1.07
RTAA*(8)	9,533 (95.33%)	−90.12	0.65
RTAA*(4)	8,883 (88.83%)	−166.40	0.48

(1.46 ms vs. 2.34 ms); however, these figures are not directly comparable: the meta-reasoner's overhead includes observation encoding and DQN inference, whereas the baseline figure reflects per-step planning time only. Despite this lower per-decision cost, RTAA*(128)'s consistent use of deep lookahead results in higher cumulative planning costs across episodes due to frequent plan invalidation, which is reflected in the more negative average reward. RTAA*(32) produces moderate performance across success rate and reward. RTAA*(8) and RTAA*(4) show progressively worse success rates (95.33% and 88.83%) and increasingly negative rewards, with RTAA*(4) performing worst overall, despite progressively lower computational overhead (0.65 ms and 0.48 ms, respectively).

All evaluations were conducted with a fixed random seed ($seed = 42$); variability across independent training runs was not assessed, and results should therefore be interpreted as representative of a single trained policy rather than as an estimate of expected performance across random initializations.

To clarify how the trained agent selects planners in response to different obstacle densities and their dynamics, we analysed its algorithm choices by density range (Figure 6.3). At very low densities (2–7%), the agent most often alternates between RTAA*(128) and RTAA*(32), with a slight preference for deeper lookahead. As density increases (7–13%), RTAA*(32) becomes the clear favourite. This dominance continues at medium density (13–18%), but RTAA*(8) starts to be selected occasionally. At high densities (18–23%), the agent uses all three algorithms more evenly, indicating increased flexibility. At very high densities (23–29%), RTAA*(8) becomes the most selected, signaling a shift toward strategies that rely on shallower lookahead and more frequent replanning. RTAA*(4) is consistently underused in all densities, indicating that very shallow lookahead is rarely advantageous under the reward structure used.

These trends indicate that the agent adapts its planner choices based on environmental complexity, choosing deeper lookahead when beneficial and switching to shallower planning as frequent changes make deep planning less effective.

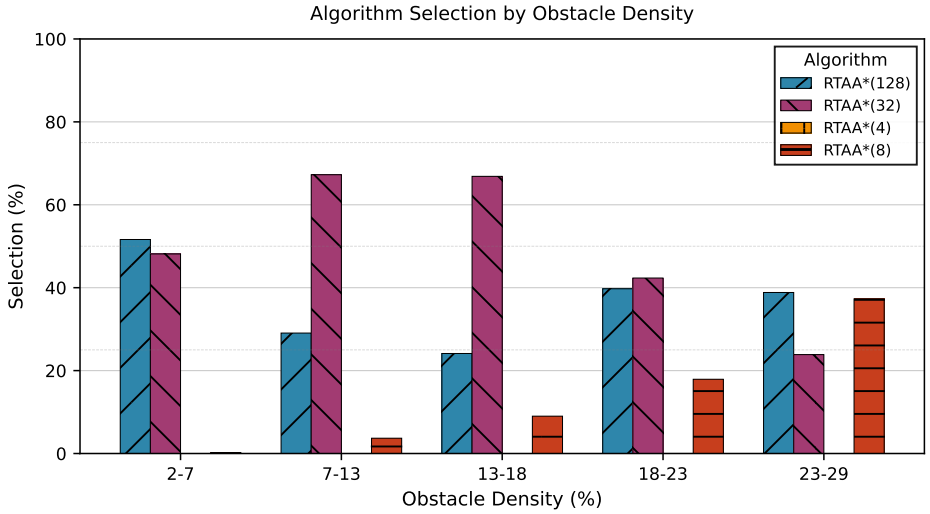


Figure 6.3: Distribution of algorithm selections by obstacle density for the RL-based meta-reasoner. Percentages reflect the proportion of each algorithm selected within each density interval.

Both results in Table 6.2 and Figure 6.3 expose an important trade-off between planning depth and adaptability in dynamic environments. In static environments, committing to a deep lookahead such as RTAA*(128) is rational, since planning cost is amortised over a stable path. However, in dynamic settings, this assumption breaks down: frequent plan invalidation forces repeated deep-lookahead replanning, accumulating costs. Conversely, RTAA*(4) avoids this by replanning cheaply, but risks local minima, yielding the worst success rate and reward. The meta-reasoner sidesteps both failure modes by switching dynamically, confirming that no fixed lookahead is consistently optimal across the range of obstacle densities and their dynamics. The meta-reasoner's modest overhead (2.34 ms per decision) yields substantial efficiency gains: the trained agent achieves a fourfold improvement in average reward over RTAA*(128) (from -61.94 to -14.50), demonstrating that the minor overhead is greatly outweighed by the benefit of avoiding unnecessary deep lookahead.

Taken together, these results show that a learning component can be invoked from within BDI reasoning without sacrificing its core strengths. The meta-reasoner preserves the symbolic, goal-directed structure of the BDI cycle while adding the flexibility to adapt how plans are computed at runtime. More broadly, these results demonstrate that

adaptive meta-reasoning over planning algorithms yields consistently better efficiency-robustness trade-offs than any of the fixed strategies evaluated: no static lookahead depth achieves both high success and low cumulative cost, whereas the learned policy does so by treating algorithm selection as a control problem rather than a one-shot prediction task.

6.5 Discussion

The results confirm that a meta-reasoner invoked at the means–end reasoning phase of the BDI deliberation cycle allows the agent not only to reason about its goals but also to revise the algorithmic strategy used to pursue them. The agent learns meaningful and consistent, environment-sensitive strategies: preferring deeper lookahead in sparse settings and switching to fast replanning when obstacles are dense and highly dynamic. Compared to fixed baselines, the adaptive agent achieves the lowest penalty and exhibits more stable performance across challenging scenarios. Because the learned component is confined to planner selection, these gains are obtained without altering the symbolic deliberation cycle, whose transparency is preserved by design.

The supervised-learning approach in Chapter 4 and the one in this chapter address complementary aspects of RQ2 but are not directly comparable empirically. Chapter 4 formulates algorithm selection as a classification problem with single-metric objectives: it trains separate models to maximize prediction accuracy for minimal path length, minimal solving time, or minimal memory consumption on static, pre-execution instances with different input spaces, whereas this chapter formulates algorithm selection as a sequential decision problem with a composite objective: a weighted reward function that simultaneously balances solution quality, computational efficiency, robustness to plan invalidation, and progress toward goals during runtime execution. The two approaches thus optimize fundamentally different objectives (independent single-metric prediction versus simultaneous multi-objective adaptation), operate under different environmental assumptions (static versus dynamic), and serve distinct roles in the thesis: Chapter 4 establishes that algorithm selection can be learned as static classification, while this chapter establishes that it can be learned as a runtime adaptive selection. Together, they validate both the predictive and adaptive aspects of RQ2 without empirical comparison between them.

The study also highlights several limitations regarding transferability. The meta-reasoner was trained and evaluated exclusively in a 10×40 two-dimensional grid with bouncing dynamic obstacles; whether the learned policy transfers to environments with

different spatial scales, topologies, or substantially different obstacle dynamics has not been tested.

The algorithm portfolio is similarly constrained: all four variants belong to the same RTAA* family and differ only in lookahead depth, excluding fundamentally different planning paradigms such as sampling-based or learning-based methods.

Finally, while the underlying architecture is domain-agnostic, the state encoding and reward function used here are tailored to pathfinding, and non-trivial adaptation would be required before applying the approach to other domains such as resource allocation or scheduling. The evaluation also focused exclusively on planner selection, leaving other BDI components unchanged.

These limitations concern the scope of the evaluation rather than the generality of the underlying architecture. The basis for expecting transferability is structural: the meta-reasoning component observes the agent's current beliefs and intentions and returns an algorithm choice, carrying no architectural dependency on pathfinding-specific representations. The weighted aggregation of performance terms-penalising computational cost, rewarding solution quality, and measuring progress toward the active intention, captures trade-offs present in any algorithm selection problem. Generalisability to domains where multiple algorithms compete on the same task is furthermore listed explicitly among the architectural design principles in Chapter 5. Domain adaptation therefore requires re-encoding the belief-intention state and re-instantiating the reward terms using domain-measurable quantities; the meta-reasoning logic and its integration into the BDI cycle remain unchanged. As a concrete illustration, scheduling under varying workload follows the same structural pattern: beliefs encode current system load and pending tasks, the active intention corresponds to the scheduling objective, and the reward penalises deadline violations and computational overhead. The empirical question of how well the trained policy transfers remains open and is identified as the most consequential direction for future work.

The learned policy is inherently sensitive to the design of the reward function. The numerical weighting of its components reflects deliberate choices motivated by the objectives of the pathfinding domain rather than an empirically optimised configuration. As detailed in Appendix B.1, these components include planning invocation costs, computational efficiency penalties, plan utilisation rewards, and terminal success bonuses. Increasing the weight assigned to computational cost would be expected to bias the learned policy toward shallower lookahead, whereas reducing this penalty would favour deeper planning. Likewise, changing the relative weighting of blocked-move penalties and terminal rewards would alter the balance between robustness and solution quality.

Although the reward coefficients were chosen to reflect the intended trade-offs of the domain, they were not systematically optimised. A comprehensive sensitivity analysis examining how different reward weightings influence the learned policy remains an important direction for future work.

Future developments will explore extending the meta-reasoning layer to other BDI components, such as intention selection or desire generation, as well as transferring the approach to domains beyond pathfinding (e.g., resource allocation or scheduling tasks). We also plan to investigate how large language models may support the offline synthesis of new planning strategies, complementing the online adaptation studied here. Overall, the results suggest a path forward for BDI agents that can flexibly adjust not only what they do, but also how they reason.

6.6 Summary

This chapter instantiated and evaluated the adaptive BDI architecture introduced in Chapter 5 within a dynamic grid-based pathfinding domain. The meta-reasoning layer was implemented using a Deep Q-Network trained to select among four RTAA* variants with differing lookahead depths, treating planner selection as a sequential decision-making problem embedded in the BDI deliberation cycle.

The experimental evaluation across 10,000 episodes demonstrated that the trained meta-reasoner outperforms all fixed-algorithm baselines, attaining the highest success rate and the least negative average reward while incurring only a modest per-decision overhead. Analysis of planner selection behavior revealed that the agent learns behaviorally interpretable, density-sensitive strategies: preferring deeper lookahead in sparse environments and switching to shallower, more reactive planning as obstacle density and volatility increase. This behavior confirms that the meta-reasoner captures meaningful environmental structure rather than converging to a trivial fixed preference.

These results address **RQ3** and **RQ4** by demonstrating that adaptive algorithm selection can be realized within a deliberative cognitive architecture without sacrificing transparency or goal-directed reasoning. The meta-reasoner enhances the BDI cycle at the means–end reasoning phase, enabling the agent to revise not only what it pursues, but how it computes the plans used to pursue it.

Several limitations of the current evaluation are acknowledged, including the restricted environment topology, the homogeneous algorithm portfolio, and the exclusive focus on planner selection within a single domain. Chapter 7 synthesises these findings in the context of the thesis's broader research questions, discusses the boundary between

empirical limitations and architectural generality, and identifies the most consequential directions for future work.

Chapter 7

Conclusion

The preceding chapters have progressively built the case for adaptive algorithm selection in autonomous agents. Chapter 3 established that algorithm performance is systematically shaped by environmental characteristics. This finding alone is not sufficient; it only motivates the question of whether such variability can be exploited. Chapter 4 answered this affirmatively, showing that environmental features can guide algorithm selection through supervised learning, but also revealed a hard boundary: static predictors cannot revise their choices once execution begins. Addressing this limitation is the core architectural contribution of Chapter 5, which embeds algorithm selection as a sequential decision-making process within the BDI deliberation cycle, grounded in the agent's live belief-intention state. Whether this design holds up empirically was the question left to Chapter 6, where the trained meta-policy consistently outperformed all fixed-algorithm baselines across varied dynamic conditions. This chapter synthesizes these findings, reflects on the limitations of the current work, and identifies the most consequential directions for future research.

7.1 Summary of Contributions

Autonomous agents operating in dynamic environments require runtime algorithm selection to effectively adapt to changing conditions and uncertainties. This thesis addressed this challenge by enabling autonomous agents to adaptively select algorithms at runtime while preserving transparency of the agent's architecture and interpretability of the conditions under which selections are made. In a dynamic pathfinding domain, we have demonstrated that incorporating learned meta-reasoning into cognitive agent architectures allows efficient runtime algorithm selection that surpasses strategies that

commit to a single algorithm throughout a task. The central contribution is a domain-agnostic framework: a deliberative BDI architecture augmented with a reinforcement learning meta-reasoner that, by design, adapts algorithm choices at runtime in domains with multiple algorithms and evolving conditions. This thesis instantiates and validates the framework in a pathfinding domain, where empirical results confirm that adaptive selection consistently outperforms fixed-algorithm baselines across varied environmental conditions.

The contribution of this thesis can be understood at three levels, which we distinguish explicitly in order to separate what is genuinely new from what is established machinery applied to a new end. *Conceptually*, we extend Rice's algorithm selection framework [44] from a single-shot, per-instance mapping to a sequential decision process, formalising runtime algorithm choice as a Markov Decision Process in which the selection is revised continuously as the environment evolves; this reformulation, which removes the static-instance assumption underlying classical algorithm selection theory, constitutes the primary theoretical contribution. *Architecturally*, we position algorithm selection as a meta-deliberative act embedded within the BDI reasoning cycle and grounded in the agent's live belief-intention state, rather than as an external pre-processing step performed once before execution; to the best of our knowledge, no prior work occupies this combination of dynamic, algorithm-level selection within a deliberative architecture. *From an engineering standpoint*, we instantiate this architecture by encoding the belief-intention state as the input to a Deep Q-Network meta-reasoner trained offline across diverse scenarios; here the constituent components are individually established, and the contribution lies in their integration and in demonstrating that the belief-intention state forms a sufficient feature representation for effective runtime selection.

These advances are delivered through three concrete contributions to adaptive algorithm selection for autonomous agents:

7.1.1 Empirical Characterization of Algorithm Performance Variability

A comprehensive evaluation of six heuristic search algorithms (D*, D* Lite, ARA*, LPA*, RTAA*, LRTA*) across varied 2D grid environments shows that no single algorithm performs best across all scenarios, and that performance depends systematically on environmental characteristics: grid size most strongly predicts memory consumption and solving time, start-goal distance correlates with path length, and obstacle density exerts a weaker influence than expected. These results quantify the cost of committing

to a single fixed algorithm and isolate the features that drive performance. Building on this, a supervised study using Random Forest classifiers showed that these features can predict the best-performing algorithm for a given metric, establishing that algorithm choice is learnable rather than arbitrary, and providing the empirical foundation for the learned, sequential selection developed in the next contribution.

7.1.2 Adaptive BDI Architecture with Learned Meta-Reasoning

A central contribution of this thesis is an extended Belief-Desire-Intention (BDI) architecture augmented with a meta-reasoning component for runtime algorithm selection. It addresses the central challenge of combining adaptive learning with deliberative transparency: the meta-reasoner is integrated into the reasoning loop so that it can be queried for which algorithm to use based on a learned policy that estimates expected long-term performance given the agent's current beliefs and intentions, while the deliberation layer itself remains transparent and the conditions of each selection interpretable. The policy is instantiated through a Deep Q-Network whose state combines environmental features from beliefs, goal-related information from intentions, and execution context, and whose reward balances solution quality, computational efficiency, and progress toward goals. Trained offline across diverse scenarios, it learns context-dependent selection without manually specified strategies, and reframes selection as a sequential decision process by adjusting choices as the environment evolves, which a one-shot predictor cannot express.

7.1.3 Empirical Demonstration of Adaptive Runtime Selection

Evaluated over 10,000 unseen dynamic episodes against four fixed-planner baselines (RTAA*(4), RTAA*(8), RTAA*(32), and RTAA*(128)), the trained meta-reasoner achieved superior overall performance in various metrics. This confirms that the architecture yields practical gains beyond theoretical feasibility, and that the improvement stems from runtime adaptivity itself rather than from any single high-performing planner.

7.2 Limitations

Several of the limitations below concern the scope of the empirical evaluation rather than the architecture itself. The meta-reasoning layer integrates into the BDI cycle

by observing the agent's current beliefs and intentions and returning an algorithm selection, with no dependency on domain-specific representations. The reward function aggregates several performance terms, in this work, solution quality, computational efficiency, and progress toward the active intention, into a single scalar via a weighted sum. The mechanism is domain-general: a new domain re-specifies which terms are relevant and how they are weighted, without changing the meta-reasoning component. The specific terms used here are particular to pathfinding and are not claimed to be the appropriate objectives for every application, nor is the weighted-sum aggregation the only possible formulation.

Domain Scope: Our empirical evaluation exclusively focuses on pathfinding in 2D grid based environments. Though our proposed architecture is domain-independent by design, empirical validation in other domains such as resource allocation, scheduling, or tool selection for LLM agents would strengthen generality claims. Also, grid-based pathfinding lacks complexities present in real-world settings, including continuous state spaces, uncertain perception, and dynamically evolving constraints that may influence the effectiveness of algorithm selection.

Limited algorithm portfolio: We considered a limited portfolio drawn from a single algorithmic family: incremental and real-time heuristic search. The empirical characterization covered six such algorithms, while the runtime selection study used RTAA* variants differing only in lookahead depth. This portfolio excludes fundamentally different paradigms such as sampling-based planners, optimization-based methods, and learning-based navigation policies.

Computational Overhead: The meta-reasoning adds computational overhead for state encoding, neural network querying, and algorithm selection before starting planning. In scenarios with short planning horizons or time-sensitive constraints, this overhead might outweigh adaptive selection benefits. Future work should quantify and identify when meta-reasoning overhead is justified versus cases when a well-chosen fixed algorithm remains sufficient.

Scalability and Computational Cost: The DQN meta-reasoner faces scalability pressures along three dimensions. Extending to multi-agent settings multiplies inference costs or introduces coordination bottlenecks if selection is centralised. Richer environments expand the state space, increasing sample complexity and risking generalisation failures. Finally, larger or more heterogeneous algorithm portfolios grow the action space, slowing DQN convergence and potentially requiring architectural changes such as hierarchical action selection. These challenges compound, meaning that scaling across all three dimensions simultaneously would likely require substantial redesign

rather than straightforward extension.

Interpretability of the Learned Policy: The BDI deliberation layer remains transparent and the conditions under which an algorithm is selected are interpretable. The learned policy is interpretable at the behavioural level: aggregate analysis of its selections reveals consistent, explicable patterns, such as a preference for shallower-lookahead variants as obstacle density and environmental dynamics increase. What it does not offer is a mechanistic account of individual decisions: the DQN's internal representations do not map onto human-interpretable concepts, so a particular selection cannot be certified or explained on a per-instance basis. Strengthening this from observed behavioural patterns to verifiable per-decision explanations is left to future work, as discussed below.

7.3 Future Research Directions

Several promising directions remain for future exploration:

Extension to Multi-Agent Settings: In this work, we focused on single-agent selection. However, extending the architecture to multi-agent settings introduces additional coordination challenges, as individual algorithm choices affect collective performance. Future work should investigate whether algorithm choice should be decentralized (i.e., each agent chooses independently) or centralized (i.e., coordinated selection across agents) for better outcomes, and how meta-reasoning policies can balance individual and team-level performance objectives.

Heterogeneous Algorithm Portfolios and Hierarchical Selection: As noted in Section 7.2, the current portfolio is drawn from a single algorithmic family, whereas path planning spans multiple paradigms. Future research should therefore investigate meta-reasoning across more diverse algorithm portfolios. This raises questions about whether a single meta-reasoner can effectively select among heterogeneous approaches or whether hierarchical meta-reasoning, meaning first selecting algorithm class, then specific variant, would be more effective.

Application to Other Domains: The most significant open direction is validating the proposed framework beyond pathfinding. The architecture is domain-agnostic by design and, in principle, applicable to settings in which multiple algorithms compete under varying conditions. Promising domains include cloud resource allocation, scheduling, tool selection in LLM-based agents, and automated planning systems. Though each would require domain-specific representations and rewards, they still preserve the core architectural principles.

Enhanced Explainability and Human Interaction: Future work should also develop mechanisms such as symbolic policy distillation, counterfactual explanations, interactive human feedback, and visualisation tools, in order to explain and refine the selection decision. This will promote trust and deployment in applications where safety is crucial.

Data and Code Availability

The datasets, generated problem instances, source code, trained model checkpoints, and evaluation outputs supporting the empirical studies in Chapters 3, 4, and 6 are publicly archived on Mendeley Data at `https://doi.org/10.17632/n3mkgmvfm3.1`.

Bibliography

[1] Zeyad Abd Algfoor, Mohd Shahrizal Sunar, and Hoshang Kolivand. A comprehensive study on pathfinding techniques for robotics and video games. *Int. J. Comput. Games Technol.*, 2015:736138:1–736138:11, 2015.

[2] Jean-Marc Alkazzi, Anthony Rizk, Michel Salomon, and Abdallah Makhoul. MAP-FASTER: A faster and simpler take on multi-agent path finding algorithm selection. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2022, Kyoto, Japan, October 23-27, 2022*, pages 10088–10093. IEEE, 2022.

[3] John R. Anderson, Daniel Bothell, Michael D. Byrne, Scott Douglass, Christian Lebiere, and Yulin Qin. An integrated theory of the mind. *Psychological Review*, 111(4):1036 – 1060, 2004. Cited by: 2417.

[4] Nafiz Arica, Aysegul Mut, Alper Yörükçü, and Kadir Alpaslan Demir. An empirical comparison of search approaches for moving agents. *Comput. Intell.*, 33(3):368–400, 2017.

[5] Amelia Badica, Costin Badica, Mirjana Ivanovic, and Dejan Mitrovic. An approach of temporal difference learning using agent-oriented programming. In *20th International Conference on Control Systems and Computer Science, CSCS 2015, Bucharest, Romania, May 27-29, 2015*, pages 735–742. IEEE, 2015.

[6] R. Peter Bonasso, R. James Firby, Erann Gat, David Kortenkamp, David P. Miller, and Marc G. Slack. Experiences with an architecture for intelligent, reactive agents. *J. Exp. Theor. Artif. Intell.*, 9(2-3):237–256, 1997.

[7] David Bond, Niels A. Widger, Wheeler Ruml, and Xiaoxun Sun. Real-time search in dynamic worlds. In Ariel Felner and Nathan R. Sturtevant, editors, *Proceedings of the Third Annual Symposium on Combinatorial Search, SOCS 2010, Stone Mountain, Atlanta, Georgia, USA, July 8-10, 2010*, pages 16–22. AAAI Press, 2010.

[8] Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley Series in Agent Technology. John Wiley & Sons, Chichester, England, October 2007.

[9] Mario Bosello and Alessandro Ricci. From programming agents to educating agents - a jason-based framework for integrating learning in the development of cognitive agents. In Louise A. Dennis, Rafael H. Bordini, and Yves Lespérance, editors, *Engineering Multi-Agent Systems (EMAS 2019)*, volume 12058 of *LNCS*, pages 175–194. Springer, Cham, 2019.

[10] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[11] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE J. Robotics Autom.*, 2(1):14–23, 1986.

[12] Mehul Damani, Zhiyao Luo, Emerson Wenzel, and Guillaume Sartoretti. Primal$_2$: Pathfinding via reinforcement and imitation multi-agent learning - lifelong. *IEEE Robotics Autom. Lett.*, 6(2):2666–2673, 2021.

[13] Edsger W. Dijkstra. A note on two problems in connexion with graphs. In Krzysztof R. Apt and Tony Hoare, editors, *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, ACM Books, pages 287–290. ACM / Morgan & Claypool, 2022.

[14] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[15] Ömer Ibrahim Erduran. Machine learning algorithms for cognitive and autonomous BDI agents. In Pascal Reuss, Viktor Eisenstadt, Jakob Michael Schönborn, and Jero Schäfer, editors, *Proceedings of the LWDA 2022 Workshops: FGWM, FGKD, and FGDB, Hildesheim (Germany), Oktober 5-7th, 2022*, volume 3341 of *CEUR Workshop Proceedings*, pages 112–123. CEUR-WS.org, 2022.

[16] João G. Faccin and Ingrid Nunes. Bdi-agent plan selection based on prediction of plan outcomes. In *2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, volume 2, pages 166–173, Singapore, 2015. IEEE.

[17] Tom Fawcett. An introduction to ROC analysis. *Pattern Recognit. Lett.*, 27(8):861–874, 2006.

[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] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6683 LNCS:507 – 523, 2011. Cited by: 2019.

[20] Princess Chinemerem Iloh. A comprehensive and comparative study of dfs, bfs, and a* search algorithms in a solving the maze transversal problem. *International Journal of Social Sciences and Scientific Studies*, 2(2):482–490, 2022.

[21] Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27(1):3 – 45, 2018. Cited by: 417; All Open Access, Bronze Open Access.

[22] Aya Kherrour, Rawlings Ntassah, Marco Robol, Marco Roveri, and Paolo Giorgini. Rl-driven meta-reasoning for bdi agents. In *Artificial Intelligence and Soft Computing – ICAISC 2026, Proceedings of the 25th International Conference, Zakopane, Poland, June 14–18, 2026*, Lecture Notes in Artificial Intelligence, pages ???–??? Springer, 2026.

[23] Aya Kherrour, Marco Robol, Marco Roveri, and Paolo Giorgini. Evaluating heuristic search algorithms in pathfinding: A comprehensive study on performance metrics and domain parameters. In Angelo Ferrando and Rafael Cardoso, editors, *Proceedings of the Third Workshop on Agents and Robots for reliable Engineered Autonomy, AREA@ECAI 2023, Krakow, Poland, 1st October 2023*, EPTCS, pages 102–112, 2023.

[24] Aya Kherrour, Marco Robol, Marco Roveri, and Paolo Giorgini. A multi-algorithm pathfinding method: Exploiting performance variations for enhanced efficiency. *Ann. Math. Artif. Intell.*, 93(4):569–588, 2025.

[25] Sven Koenig. A comparison of fast search methods for real-time situated agents. In *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), 19-23 August 2004, New York, NY, USA*, pages 864–871. IEEE Computer Society, 2004.

[26] Sven Koenig and Maxim Likhachev. D*lite. In Rina Dechter, Michael J. Kearns, and Richard S. Sutton, editors, *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada*, pages 476–483. AAAI Press / The MIT Press, 2002.

[27] Sven Koenig and Maxim Likhachev. Real-time adaptive a*. In Hideyuki Nakashima, Michael P. Wellman, Gerhard Weiss, and Peter Stone, editors, *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), Hakodate, Japan, May 8-12, 2006*, pages 281–288, New York, USA, 2006. ACM.

[28] Sven Koenig, Maxim Likhachev, and David Furcy. Lifelong planning A*. *Artif. Intell.*, 155(1-2):93–146, 2004.

[29] Sven Koenig and Xiaoxun Sun. Comparing real-time and incremental heuristic search for real-time situated agents. *Auton. Agents Multi Agent Syst.*, 18(3):313–341, 2009.

[30] Richard E Korf. Real-time heuristic search. *Artificial intelligence*, 42(2-3):189–211, 1990.

[31] John E. Laird, Allen Newell, and Paul S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1 – 64, 1987. Cited by: 1701.

[32] Sharmad Rajnish Lawande, Graceline Jasmine, Jani Anbarasi, and Lila Iznita Izhar. A systematic review and analysis of intelligence-based pathfinding algorithms in the field of video games. *Applied Sciences*, 12(11):5499, 2022.

[33] Maxim Likhachev, Geoffrey J. Gordon, and Sebastian Thrun. Ara*: Anytime a* with provable bounds on sub-optimality. In Sebastian Thrun, Lawrence K. Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16 [Neural Information Processing Systems, NIPS 2003, December 8-13, 2003, Vancouver and Whistler, British Columbia, Canada]*, pages 767–774. MIT Press, 2003.

[34] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[35] 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. Cited by: 28310.

[36] Sara Lutami Pardede, Fadel Ramli Athallah, Yahya Nur Huda, and Fikri Dzulfiqar Zain. A review of pathfinding in game development. *CEPAT] Journal of Computer Engineering: Progress, Application and Technology*, 1(01):47, 2022.

[37] Marzie Parooei, Mehdi Tale Masouleh, and Ahmad Kalhor. MAP3F: a decentralized approach to multi-agent pathfinding and collision avoidance with scalable 1d, 2d, and 3d feature fusion. *Intell. Serv. Robotics*, 17(3):401–418, 2024.

[38] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12:2825–2830, 2011.

[39] Silvester Handy Permana, Ketut Yogha Bintoro, Budi Arifitama, Ade Syahputra, et al. Comparative analysis of pathfinding algorithms a*, dijkstra, and bfs on maze runner game. *IJISTECH (International J. Inf. Syst. Technol*, 1(2):1, 2018.

[40] Szymon Pulawski, Hoa Khanh Dam, and Aditya Ghose. Bdi-dojo: developing robust bdi agents in evolving adversarial environments. In *ACSOS 2021 Companion*, pages 257–262, Washington, DC, USA, 2021. IEEE.

[41] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *J. Mach. Learn. Res.*, 22:268:1–268:8, 2021.

[42] Anand S. Rao and Michael P. Georgeff. Bdi agents: From theory to practice. In *Proceedings of the First International Conference on Multiagent Systems*, pages 312–319, San Francisco, California, USA, 1995. The MIT Press.

[43] Jingyao Ren, Vikraman Sathiyanarayanan, Eric Ewing, Baskin Senbaslar, and Nora Ayanian. MAPFAST: A deep algorithm selector for multi agent path finding using shortest path embeddings. In Frank Dignum, Alessio Lomuscio, Ulle

Endriss, and Ann Nowé, editors, *AAMAS '21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021*, pages 1055–1063. ACM, 2021.

[44] John R. Rice. The algorithm selection problem. *Adv. Comput.*, 15:65–118, 1976.

[45] Stuart Russell and Eric Wefald. Principles of metareasoning. *Artificial Intelligence*, 49(1-3):361 – 395, 1991. Cited by: 195.

[46] Devon Sigurdson, Vadim Bulitko, Sven Koenig, Carlos Hernández, and William Yeoh. Automatic algorithm selection in multi-agent pathfinding. *CoRR*, abs/1906.03992, 2019.

[47] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[48] Deepak Singh, Sebastian Sardiña, and Lin Padgham. Extending bdi plan selection to incorporate learning from experience. *Robotics and Autonomous Systems*, 58(9):1067–1075, 2010.

[49] Dhirendra Singh, Sebastian Sardiña, Lin Padgham, and Geoff James. Integrating learning into a BDI agent for environments with changing dynamics. In Toby Walsh, editor, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 2525–2530. IJCAI/AAAI, 2011.

[50] Kate A. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 41(1), 2008. Cited by: 444.

[51] Anthony Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the 1994 International Conference on Robotics and Automation, San Diego, CA, USA, May 1994*, pages 3310–3317. IEEE Computer Society, 1994.

[52] Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Roman Barták, and Eli Boyarski. Multi-agent pathfinding: Definitions, variants, and benchmarks. In Pavel Surynek and William Yeoh, editors, *Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019, Napa, California, 16-17 July 2019*, pages 151–158. AAAI Press, 2019.

[53] Nathan R Sturtevant. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):144–148, 2012.

[54] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998.

[55] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

[56] Qian Wan, Wei Liu, Longlong Xu, and Jingzhi Guo. Extending the BDI model with q-learning in uncertain environment. In *Proceedings of the 2018 International Conference on Algorithms, Computing and Artificial Intelligence, ACAI 2018, Sanya, China, December 21-23, 2018*, pages 33:1–33:6. ACM, 2018.

[57] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers Comput. Sci.*, 18(6):186345, 2024.

[58] Marco A. Wiering, Maikel Withagen, and Madalina M. Drugan. Model-based multi-objective reinforcement learning. 2014. Cited by: 41; All Open Access, Green Open Access.

[59] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Satzilla: Portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research*, 32:565 – 606, 2008. Cited by: 737; All Open Access, Gold Open Access, Green Open Access.

[60] Imants Zarembo and Sergejs Kodors. Pathfinding algorithm efficiency analysis in 2d grid. In *ENVIRONMENT. TECHNOLOGIES. RESOURCES. Proceedings of the International Scientific and Practical Conference*, volume 2, pages 46–50, 2013.

Appendix A

MAPF Benchmark Details

Table A.1 lists the 32 maps from the MAPF benchmark suite [52] used in the empirical study of Chapter 4. The map `orz900d` was excluded due to its excessive computational cost. The maps span diverse categories including empty grids, mazes, city environments, and game-inspired layouts, providing a comprehensive testbed for evaluating path-finding algorithms.

Table A.1: The 32 MAPF benchmark maps used in the study of Chapter 4

Map name	**Category**	**Height**	**Width**
Berlin_1_256	City	256	256
Boston_0_256	City	256	256
Paris_1_256	City	256	256
brc202d	Game	481	530
den312d	Game	81	65
den520d	Game	257	256
✓empty-16-16	Empty	16	16
empty-32-32	Empty	32	32
empty-48-48	Empty	48	48
ht_chantry	Game	141	162
ht_mansion_n	Game	270	133
lak303d	Game	194	194
lt_gallowstemplar_n	Game	180	251
maze-32-32-2	Maze	32	32
maze-32-32-4	Maze	32	32
maze-128-128-1	Maze	128	128
maze-128-128-2	Maze	128	128
maze-128-128-10	Maze	128	128
ost003d	Game	194	194
random-32-32-10	Random	32	32
random-32-32-20	Random	32	32
random-64-64-10	Random	64	64
random-64-64-20	Random	64	64
room-32-32-4	Room	32	32
room-64-64-8	Room	64	64
room-64-64-16	Room	64	64
w_woundedcoast	Game	578	642
warehouse-10-20-10-2-1	Warehouse	63	161
warehouse-10-20-10-2-2	Warehouse	84	170
warehouse-20-40-10-2-1	Warehouse	123	321
warehouse-20-40-10-2-2	Warehouse	164	340

Appendix B

Experimental Setup Details

B.1 Reward Function Coefficients

Table B.1 reports the numerical values of all active components of the reward function used during DQN training (Section 6.4). All reward values are computed inline within the environment step function. Terminal rewards are applied once at episode conclusion and are set symmetrically at ±50 to provide a strong outcome signal relative to the per-step penalties.

Table B.1: Reward function component values used in experimental evaluation.

Component	**Description**	**Value**
Goal reached	Terminal reward for reaching goal	$+50$
Episode failure	Terminal penalty for timeout/failure	-50
Planning invocation	Fixed penalty per planning or replan call	-5
Planning failure	Penalty when planner returns no valid path	-10
Nodes expanded	Proportional penalty per expanded node	$-0.01 \times n$
Lookahead efficiency	Reward for fraction of lookahead used	$+0.01 \times n/n_{\max}$
Plan utilisation	Reward for fraction of plan steps executed	$+k_{\text{exec}}/k_{\text{plan}}$
Blocked move	Penalty when a planned move is blocked	-2

The relative magnitudes of these components encode a deliberate priority ordering rather than arbitrary tuning. The terminal outcomes (±50) sit an order of magnitude above the per-step terms, so that reaching or failing to achieve the goal dominates the return and task success remains the primary objective, while the remaining terms shape *how* that outcome is achieved. Among the per-step terms, a planning failure (-10) is penalised more heavily than a routine planning invocation (-5), since a planner that returns no valid path incurs the cost of deliberation without yielding progress.

The node-expansion penalty is kept small per node (-0.01) so that it accumulates into a meaningful cost only when a planner expands many nodes, allowing the reward to discriminate between shallow and deep lookahead by their computational footprint without overwhelming the discrete penalties. The small positive lookahead-efficiency and plan-utilisation terms counterbalance the invocation penalty, rewarding productive use of the computed plans rather than discouraging planning outright. Finally, the blocked-move penalty (-2) discourages committing to plans that are quickly invalidated by moving obstacles, while remaining modest because occasional blocked moves are unavoidable in a dynamic environment.

B.2 DQN Training Diagnostics

Figure B.1 presents the training diagnostics for the DQN meta-reasoner over approximately 5×10^6 environment steps. The best-performing checkpoint, selected by the evaluation callback during training, was used for all results reported in Section 6.4.

Panel (a): Rollout reward. The smoothed mean episode reward begins at approximately -25 and improves steadily, converging to around -6 by the end of training, indicating consistent policy improvement throughout the full training run.

Panel (b): Evaluation reward. The periodic evaluation reward (computed every 10,000 steps over 10 held-out episodes) exhibits high variance throughout training, with the smoothed curve oscillating between approximately -50 and 0 and occasionally dipping to -75. Despite this variance, an upward trend is visible over the second half of training. Raw evaluation points below -200 are clipped in the display.

Panel (c): TD loss. The smoothed temporal-difference loss drops sharply from approximately 10 to 5 within the first 500,000 steps and stabilises for the remainder of training. The narrow oscillation band ($[4.0, 5.0]$) observed thereafter indicates that the Q-Network reached a stable approximation of the action-value function well before evaluation.

Panel (d): Exploration rate. The ε-greedy exploration rate was configured to decay linearly from 1.0 to 0.05 over 33% of the total training budget. Since the episode limit was reached before the decay horizon, the exploration rate did not reach its target minimum, ending at approximately 0.68. The agent therefore maintained substantial exploration throughout training. The complete configuration is given in Table 6.1. Taken together, the four panels confirm that the DQN converged to a stable policy prior to the evaluation reported in Section 6.4.

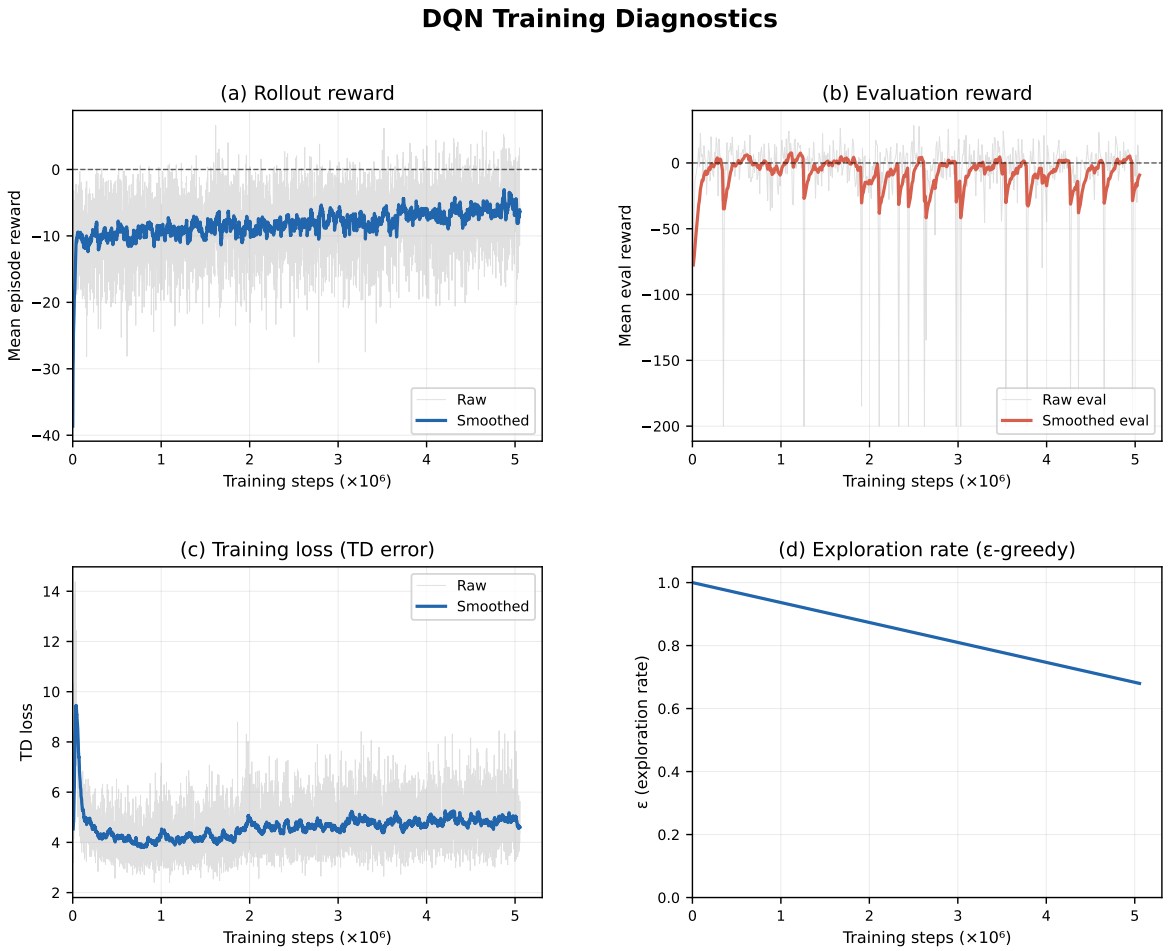


Figure B.1: DQN training diagnostics over approximately 5×10^6 environment steps. (a) Mean rollout reward per episode (smoothed). (b) Mean periodic evaluation reward (smoothed; raw points below -200 clipped). (c) Temporal-difference (TD) loss (smoothed). (d) ε-greedy exploration rate (decay incomplete at training termination, final $\varepsilon \approx 0.68$).

B.3 Training Environment

Figure B.2 illustrates the training environment used for all experiments. The grid measures 10×40 cells with a fixed start position at $(0, 0)$ and a fixed goal at $(9, 39)$. At each episode, between 1 and 130 dynamic obstacles are placed randomly across free cells, avoiding the agent, start, and goal positions. Each obstacle is initialised with a random direction, a speed drawn from $\{0, 1, 2\}$ cells per step, and a random per-axis acceleration drawn from $\{-1, 0, 1\}$. Obstacle positions are updated according to second-order dynamics: at each update event, velocity is incremented by acceleration and clamped to $[-2, 2]$; if the resulting target cell is occupied or out of bounds, the

velocity is reversed. Update events occur every f micro-steps, where f is sampled uniformly from $[3, 8]$ at the start of each episode.

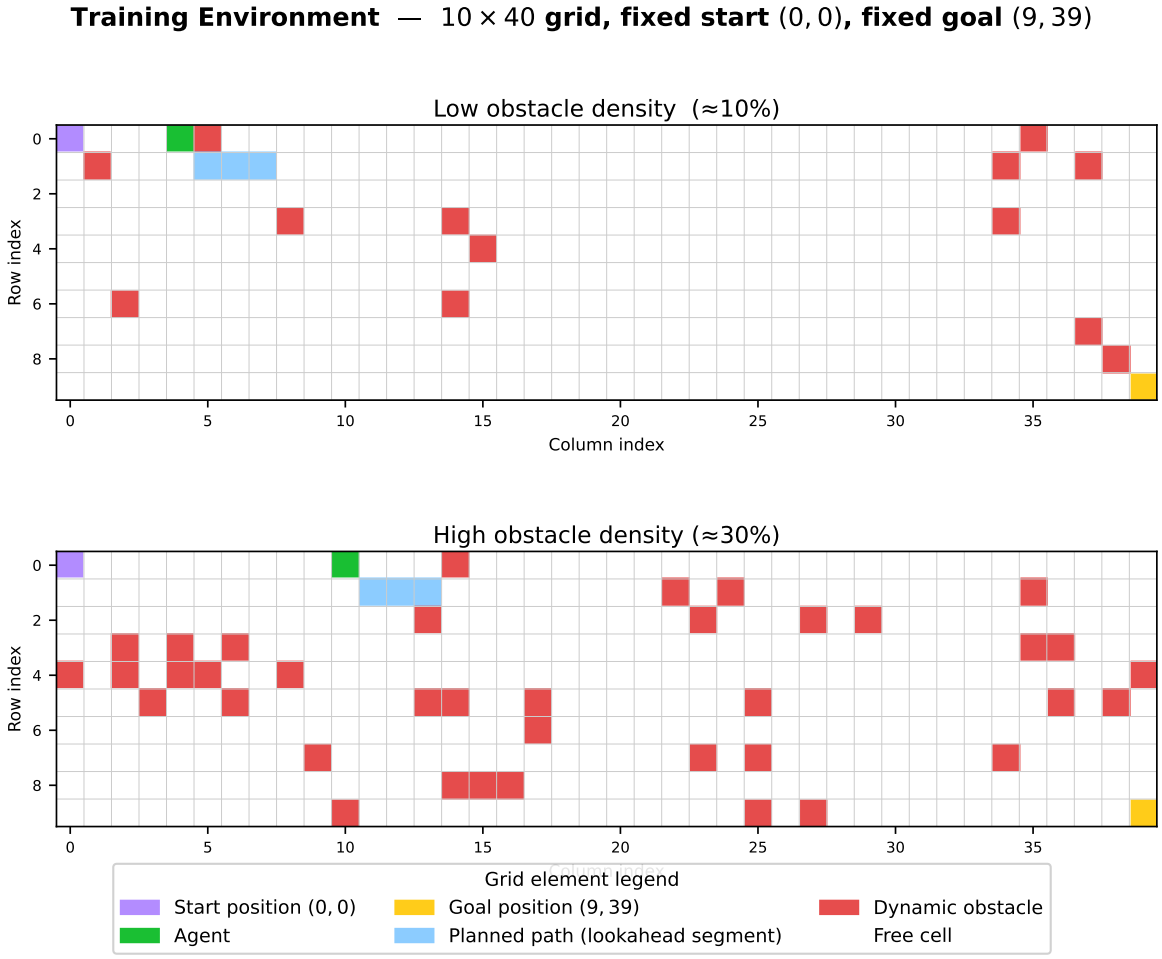


Figure B.2: Training environment snapshots at two obstacle density levels. The grid is 10×40 cells. **Purple**: fixed start $(0, 0)$. **Green**: agent current position. **Yellow**: fixed goal $(9, 39)$. **Light blue**: planned path segment (lookahead). **Red**: dynamic obstacles. **White**: free cells. Obstacles move stochastically at each step; their count and movement probability vary across episodes to produce a range of environment change rates (ECR).