

# A Hybrid Hierarchical Game Agent Framework for Dota 2

*Zhenglan Chen*

A dissertation submitted in partial fulfilment  
of the requirements for the degree of  
**Bachelor of Science**  
of the  
**University of Aberdeen.**



SCNU Joint Institute

2026

# Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed:

Date: 2026

# Abstract

An expansion of the title and contraction of the thesis.

# Acknowledgements

Much stuff borrowed from elsewhere

# Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
1.1	Context and Motivation . . . . .	6
1.2	Overview of Dota 2 . . . . .	7
1.3	Research Problem . . . . .	8
1.4	Proposed Solution . . . . .	9
1.5	Scope of Study . . . . .	9
1.6	Significance of Study . . . . .	10
1.7	Thesis Structure . . . . .	11
<b>2</b>	<b>Related Work</b>	<b>12</b>
2.1	Hierarchical Reinforcement Learning and Temporal Abstraction . . . . .	12
2.2	Game AI and RL Applications in RTS/MOBA Games . . . . .	13
2.3	Offline Reinforcement Learning for Itemization . . . . .	14
2.4	Language Models and Agent Frameworks . . . . .	16
2.5	Evaluation Metrics in Game AI . . . . .	17
<b>3</b>	<b>Methodology</b>	<b>19</b>
3.1	Literature Synthesis and Methodological Gaps . . . . .	19
3.2	API Integration and Prototyping Workflow . . . . .	20
3.3	Architectural Design of the Three-Level Framework . . . . .	21
3.3.1	Tactical Layer: Behavior-Tree Execution and Micro-Control . . . . .	22
3.3.2	Item-Buying Layer: Offline Reinforcement Learning Formulation . . . . .	22
3.3.3	Strategic Layer: Language-Model Reasoning and Tool Integration . . . . .	23
3.3.4	Orchestration Runtime: Cross-Module Communication and Safety Arbitration . . . . .	23
3.4	Data Pipeline and Baseline Construction . . . . .	24
3.5	Methodological Constraints and Validation Framework . . . . .	25
<b>4</b>	<b>Frequently asked questions</b>	<b>26</b>
4.1	References . . . . .	26
4.2	Figures . . . . .	26
4.3	Frequently used symbols . . . . .	27

## Chapter 1

# Introduction

### 1.1 Context and Motivation

Real-time strategy (RTS) and multiplayer online battle arena (MOBA) games have become important benchmark domains for artificial intelligence because they combine real-time control, delayed reward, adversarial multi-agent interaction, partial observability, and large combinatorial decision spaces in a single environment. In contrast to turn-based board games, competent play in these domains requires an agent to react continuously while also reasoning over long temporal horizons, coordinating with teammates, adapting to hidden information, and balancing local combat execution with global objectives. Landmark systems such as AlphaStar for StarCraft II and OpenAI Five for Dota 2 demonstrated that learning-based agents can achieve elite performance in these settings, but they also made clear that such success typically depends on enormous engineering effort, large-scale distributed infrastructure, and substantial training compute. These properties make RTS/MOBA games both scientifically valuable and practically demanding as research environments for autonomous agents (Berner et al., 2019; Vinyals et al., 2019)

Within this broader landscape, reinforcement learning (RL) provides a natural formalism for sequential decision-making under uncertainty. In the standard RL framework, an agent interacts with an environment, receives scalar reward, and improves its policy over time through experience (?). However, long-horizon environments such as MOBAs expose a central weakness of flat decision policies: many strategically important decisions only reveal their consequences after dozens or hundreds of low-level actions. Hierarchical reinforcement learning (HRL) addresses this issue by introducing temporal abstraction. Foundational work on the options framework formalized temporally extended actions as closed-loop policies that operate over multiple time steps (?). MAXQ further showed how complex tasks can be decomposed into subtasks with structured value functions (?). Later, FeUdal Networks demonstrated how managers and workers operating at different time scales can improve long-term credit assignment and induce meaningful sub-policies (?). Together, these lines of work suggest that hierarchical decomposition is not merely an implementation convenience, but a principled response to long-horizon decision problems.

At the same time, recent progress in AI suggests that no single paradigm is sufficient for every layer of complex game behavior. Large-scale end-to-end self-play can produce strong policies, as shown by OpenAI Five, but such approaches are difficult to reproduce in ordinary research settings (Berner et al., 2019). Offline RL offers a complementary route by learning from previously collected interaction data rather than expensive online exploration (?). In parallel, large language

models (LLMs) have shown promise as high-level reasoning modules that can interleave symbolic planning, tool use, and environment interaction, as illustrated by ReAct, Toolformer, and Voyager (???). For Dota 2, this combination is particularly attractive: the game exposes a programmable bot interface through Lua scripting, replay-derived datasets are available through resources such as OpenDota, and the strategic layer of play is naturally expressible in language-like abstractions such as lane pressure, timing windows, objective control, and item timing (??).

This thesis is motivated by the observation that a practical academic Dota 2 agent should not attempt to solve the entire game using one monolithic policy. Instead, it should exploit the structure of the domain: high-frequency mechanical actions can be handled by deterministic tactical controllers, medium-horizon economic decisions can be learned from replay data, and slower strategic reasoning can be delegated to a planning module that operates over abstract game concepts. Such a design aims to preserve the strengths of learning-based systems while improving modularity, interpretability, reproducibility, and engineering feasibility relative to fully end-to-end approaches.

## 1.2 Overview of Dota 2

Dota 2 is a commercial competitive game developed and published by Valve. It is commonly categorized as a MOBA and is also described by Valve as an “action RTS.” The game pits two teams of five players against each other, with each player controlling a single hero selected from a roster of over one hundred heroes. The primary objective is to destroy the opposing team’s Ancient, the central structure located in the enemy base, while defending one’s own (??).

Although the win condition is simple, the route to victory is strategically rich. The map is organized around lanes, bases, towers, barracks, neutral camps, rune locations, and other objective-bearing regions. Valve’s developer documentation for Dota-style maps identifies core map entities such as Ancients, towers, barracks, fountains, shops, runes, neutral camps, and Roshan, reflecting the structural components that shape standard play (?). Recent major updates have further expanded map complexity. In patch 7.33, Valve introduced a reworked map with Twin Gates, two Roshan pits, and Tormentor objectives, and later patch notes continued to adjust Roshan and Tormentor mechanics as part of the active objective layer (??). Accordingly, modern Dota 2 requires agents to reason not only about lanes and buildings, but also about rotating map objectives and changing terrain affordances.

Each player interacts with the game through a hero that possesses unique abilities, attributes, and combat characteristics. Valve emphasizes that the hero pool is “massive and limitlessly diverse,” and that heroes can express widely different tactical patterns through spells, attack types, and ultimate abilities (?). Hero progression is tightly coupled with itemization and skill choices: the official Hero Builds system explicitly organizes in-game guidance around which abilities to level up and when, together with which items best suit the hero (?). Importantly, Valve also notes that Dota 2 does not rigidly constrain heroes to a single role; rather, “any hero can fill multiple roles,” and item choices allow players to adapt to the needs of a particular match (?). This flexibility is a major source of complexity for autonomous agents because it enlarges the space of viable builds, tactics, and team compositions.

From a team-play perspective, Dota 2 is not just a 5v5 combat simulator but a coordination problem involving differentiated economic and tactical responsibilities. Prior research on role identification in Dota 2 argues that player roles materially affect how performance should be interpreted, and that treating all heroes or players with the same metrics obscures important strategic distinctions (?). In other words, a farming core, a roaming initiator, and a defensive support do not optimize the same local objectives, even if all ultimately contribute to destroying the enemy Ancient. This matters for agent design because good behavior cannot be reduced to isolated combat skill; it also requires lane allocation, farm prioritization, timing-sensitive rotations, and objective-oriented team play (?).

For autonomous control, Dota 2 is especially interesting because Valve exposes a server-side Lua bot scripting interface. Through this interface, scripts can query authorized game-state information and issue commands directly to controlled units rather than relying on screen scraping or input emulation. At the same time, the API explicitly enforces game-realistic information constraints: scripts cannot query units hidden in the fog of war and cannot issue commands to units outside their control (?). This makes the environment attractive for AI research: it is programmable and information-rich, yet still preserves the partial observability that is fundamental to human play.

### 1.3 Research Problem

This thesis addresses three interrelated challenges that arise when building autonomous agents for Dota 2.

First, the game exhibits extreme decision horizons. A single match can last tens of minutes, and strategic choices made in the opening lane phase may determine the outcome much later through accumulated gold, experience, vision control, map pressure, and objective tempo. OpenAI explicitly highlighted Dota 2 as a domain with long time horizons, imperfect information, and complex state-action spaces (Berner et al., 2019). A flat controller that chooses only immediate actions may handle local combat competently while still failing to pursue economically and strategically coherent long-term plans.

Second, the game is partially observable and information asymmetric. Because of the fog of war, an agent must act under uncertainty about enemy positions, rotations, smoke movements, and hidden objectives. The bot API mirrors this constraint by restricting access to unseen units (?). As a result, the agent cannot simply compute globally optimal responses from full state information. Instead, it must reason from incomplete observations, game context, and probabilistic expectations about enemy behavior.

Third, competent play requires coordination across tactical and strategic levels. In Dota 2, low-level execution includes movement, attack timing, ability casting, retreating, targeting, and positioning. Medium-level economic decisions include farming patterns and item purchases. High-level strategic decisions include lane assignments, power-spike timing, tower pressure, Roshan control, and team movement patterns. These layers are tightly coupled. For example, an item purchase changes the feasible tactical repertoire; a strategic push call changes whether lane creeps or jungle camps should be prioritized; and role-dependent expectations alter how success should be measured (????). Designing an agent that coordinates these levels without collapsing them into an intractable monolithic policy is the central problem considered in this work.



## 1.4 Proposed Solution

To address these challenges, this thesis proposes a three-level hybrid hierarchical game agent framework for Dota 2. The framework separates the game into decision layers with different temporal resolutions and computational requirements.

At the tactical level, a Lua-based controller implements behavior-tree-like decision logic for immediate combat and movement behaviors. This layer is responsible for executing micro-actions robustly and with low latency, leveraging the official bot scripting interface for direct control (?). At the item-buying level, the framework uses offline reinforcement learning trained on replay-derived data to learn purchase policies from historical match trajectories. This design is motivated by the suitability of offline RL for static datasets (?), together with algorithms such as DQfD, which combines demonstration learning with temporal-difference updates (?). and CQL, which learns conservative value estimates for offline policy learning (?). At the strategy level, a large language model produces high-level directives—such as lane priorities, objective timing suggestions, or team posture recommendations—using structured prompts and tool-like interaction patterns inspired by recent LLM-agent research including ReAct, Toolformer, and Voyager(??).

The key idea is that these three layers are complementary rather than redundant. Tactical control benefits from explicit, debuggable rules and low-latency execution. Itemization benefits from data-driven learning over large replay corpora, where historical expert behavior can be exploited without unsafe online exploration. Strategic planning benefits from a model capable of operating over semantic abstractions, textual memory, and structured reasoning steps. By assigning each layer the kind of reasoning it handles best, the framework seeks to achieve a better trade-off among performance, interpretability, and implementation practicality than either purely scripted bots or fully end-to-end neural agents.

## 1.5 Scope of Study

The scope of this research is deliberately bounded to ensure methodological rigor, reproducibility, and a focused investigation into hierarchical decision-making within the Dota 2 environment. Primarily, the study concentrates on the design, integration, and empirical evaluation of a single autonomous agent operating within a team-based match. While Dota 2 is inherently a multi-agent environment, this work isolates the decision pipeline of one controlled hero to examine how hierarchical decomposition—spanning tactical execution, itemization, and strategic planning—can be systematically engineered and validated. The agent interacts with the game exclusively through Valve’s official server-side Lua bot scripting interface, which provides structured game-state information and deterministic command execution. Consequently, visual perception, screen-scraping, or pixel-based control are excluded from this study; the agent operates on symbolic and semi-structured state representations provided by the game engine.

Furthermore, the research does not address the pre-game phases of matchmaking, hero drafting, or team composition optimization. The agent is instantiated with a predefined hero and role assignment at match initialization. The item-buying module is constrained to offline reinforcement learning paradigms, leveraging historical replay datasets rather than engaging in costly online exploration or self-play loops. This boundary is intentional, aligning with the study’s emphasis on sample efficiency, safety, and the exploitation of expert trajectories. Similarly, the strategic layer is

scoped to high-level directive generation via large language models, focusing on textual reasoning, tool-integration patterns, and structured prompt engineering rather than end-to-end neural policy training for macro-decisions.

Experimentally, the evaluation is confined to a curated set of hero-role combinations, standardized map configurations, and controlled opponent baselines (e.g., scripted bots and intermediate AI opponents). The study prioritizes interpretability, modularity, and cross-layer interoperability over raw competitive dominance against elite human or fully optimized self-play agents. The framework assumes a stable network environment and deterministic game physics within the bot API sandbox, meaning that network latency variations, client-side rendering artifacts, or anti-cheat interference are outside the analytical scope. Additionally, while the orchestration runtime handles inter-module communication and safety checks, the research does not extend to distributed multi-agent training infrastructures, cloud-based scaling, or real-time adversarial fine-tuning during live matches. These boundaries ensure that the research remains tractable within academic resource constraints while still addressing the core challenges of long-horizon decision-making, partial observability, and hierarchical coordination in a complex MOBA environment.

## 1.6 Significance of Study

The significance of this research lies in its demonstration that complex, real-time decision-making environments can be effectively navigated through a modular, hybrid architecture rather than relying exclusively on monolithic, end-to-end learning pipelines. By explicitly separating tactical execution, economic optimization, and strategic planning into interoperable layers, this work challenges the prevailing assumption that supreme performance in MOBA environments necessitates massive-scale self-play and billion-parameter neural networks. Instead, it establishes a reproducible paradigm that leverages the complementary strengths of classical control logic, offline reinforcement learning, and large language models, offering a more accessible and interpretable pathway for academic research in game AI.

From a methodological perspective, the study advances the practical application of hierarchical reinforcement learning by grounding temporal abstraction in a concrete, multi-scale implementation. The tactical layer provides deterministic, low-latency micro-control, ensuring baseline competence without the instability of continuous policy updates. The itemization layer demonstrates how offline reinforcement learning can safely and efficiently extract economically sound policies from historical datasets, circumventing the exploration hazards and computational costs of online training. The strategic layer illustrates how contemporary language models can function as high-level reasoning engines, capable of synthesizing game state, role-specific objectives, and temporal constraints into actionable directives. Crucially, the integration of these layers via a robust orchestration runtime highlights the engineering feasibility of heterogeneous AI systems, addressing long-standing challenges in cross-language communication, latency management, and fault tolerance.

Beyond the immediate domain of Dota 2, this work carries broader implications for the development of autonomous agents in complex, partially observable environments. The architectural principles explored here—modular decomposition, safe offline learning, and tool-augmented strategic reasoning—are directly transferable to robotics, autonomous logistics, and multi-agent

coordination systems where interpretability, safety, and resource efficiency are paramount. In domains such as surgical robotics or autonomous vehicle fleets, the ability to delegate high-frequency control to verified rule-based systems, optimize mid-term resource allocation via offline data, and employ high-level reasoning modules for adaptive planning mirrors the hierarchical structure proposed in this thesis. By providing a fully documented, open-architecture framework, this study also contributes to the growing demand for reproducible AI research, enabling independent validation, iterative improvement, and community-driven extensions.

Furthermore, the research addresses a critical gap in the literature concerning the operationalization of LLMs in real-time, closed-loop environments. While prior work has extensively documented LLM capabilities in text-based games, puzzle solvers, or slow-turn simulations, their application in high-frequency, partially observable, adversarial settings remains underexplored. This study bridges that gap by demonstrating how structured prompting, constrained tool-use, and latency-aware integration can transform LLMs from passive reasoning modules into active strategic directors. The empirical validation of this approach, alongside rigorous ablation studies comparing architectural variants, provides actionable insights for both AI researchers and game developers seeking to integrate modern machine learning paradigms into interactive systems. Ultimately, the research bridges the gap between theoretical hierarchical decision-making models and practical, deployable game AI, offering a scalable blueprint that balances performance, transparency, and engineering practicality.

## 1.7 Thesis Structure

The remainder of this thesis is organized as follows. Chapter 2 reviews related work in hierarchical reinforcement learning, game AI for RTS and MOBA environments, offline reinforcement learning for decision-making from replay data, and LLM-based agent frameworks. Chapter 3 presents the methodology, including literature study, prototyping process, system design, and the overall architecture of the proposed three-level agent. Chapter 4 describes the implementation details of the tactical controller, the item-buying RL pipeline, the LLM strategy module, and the orchestration runtime used to integrate them. Chapter 5 reports experiments and results, including baseline comparisons, ablation studies, and robustness analyses. Chapter 6 discusses the implications of the results, the limitations of the system, and its relation to prior work. Finally, Chapter 7 concludes the thesis by summarizing contributions and outlining directions for future research.

## Chapter 2

# Related Work

The development of autonomous agents for complex, partially observable, and long-horizon environments has evolved rapidly over the past two decades. This chapter surveys the foundational and contemporary literature that informs the design of a hybrid hierarchical game agent framework for Dota 2. The review is organized into five thematic areas: hierarchical reinforcement learning and temporal abstraction, game AI and reinforcement learning applications in RTS/MOBA domains, offline reinforcement learning for sequential decision-making from static datasets, large language models and agent reasoning frameworks, and evaluation methodologies for game AI. By synthesizing these research strands, this chapter identifies the theoretical gaps, methodological limitations, and engineering constraints that motivate the proposed three-level architecture.

## 2.1 Hierarchical Reinforcement Learning and Temporal Abstraction

Hierarchical reinforcement learning (HRL) emerged as a direct response to the limitations of flat Markov Decision Process (MDP) formulations in environments where optimal behavior requires reasoning over multiple time scales. In standard RL, an agent maximizes expected cumulative reward by selecting primitive actions at each time step. However, in domains with long decision horizons, sparse rewards, and compositional task structures, flat policies suffer from severe credit assignment problems, poor sample efficiency, and difficulty in generalizing across subtasks. Temporal abstraction addresses these issues by allowing agents to operate over temporally extended actions, or options, which encapsulate policies, initiation sets, and termination conditions (?).

The options framework, formalized by Sutton, Precup, and Singh, established a rigorous mathematical foundation for hierarchical decision-making by demonstrating that value functions could be decomposed across temporal layers without violating the Bellman optimality conditions (?). This framework showed that high-level controllers could select among options that themselves executed sequences of low-level actions, effectively compressing the decision tree and enabling more efficient exploration. Building on this, Dietterich's MAXQ value function decomposition introduced a structured representation of hierarchical tasks as directed acyclic graphs of subtasks, where each subtask maintained its own value function and reward signal (?). MAXQ demonstrated that complex sequential decision problems could be broken down into recursively solvable components, with explicit handling of shared subroutines and state abstraction.

While early HRL approaches relied on hand-crafted hierarchies and symbolic task definitions, the advent of deep reinforcement learning enabled data-driven discovery of hierarchical

structures. Kulkarni et al. introduced h-DQN, which trained separate deep networks for high-level goal selection and low-level policy execution, demonstrating that hierarchical decomposition could improve sample efficiency in navigation and control tasks (?). Similarly, Nachum et al. proposed HIRO, a theoretically grounded algorithm that learned hierarchical policies with provable guarantees on subgoal feasibility and value decomposition, reducing the instability often associated with multi-level policy optimization (?). These methods highlighted the importance of goal-conditioned policies and the need for stable gradient flow across hierarchical layers.

A significant milestone in deep HRL was the introduction of FeUdal Networks by Vezhnevets et al., which drew inspiration from feudal systems by separating a manager network that sets directional goals from a worker network that executes primitive actions to achieve those goals (?). By operating at different temporal frequencies and using intrinsic motivation signals, FeUdal Networks achieved improved long-term credit assignment and emergent subtask specialization in continuous control environments. Subsequent work extended these ideas with attention mechanisms, memory augmentation, and multi-agent coordination protocols, demonstrating that hierarchical architectures could scale to increasingly complex domains (??).

Despite these advances, applying HRL to real-time, partially observable games like Dota 2 introduces unique challenges. First, the non-stationarity induced by adversarial opponents and dynamic team compositions complicates the stability of hierarchical value functions. Second, the lack of explicit subtask boundaries in MOBA gameplay makes it difficult to define clean initiation and termination conditions for options. Third, the computational overhead of training multiple policy networks in parallel often exceeds the resources available to academic research groups. Consequently, many recent studies have shifted toward hybrid approaches that combine learned hierarchical policies with hand-designed control layers, leveraging the strengths of both paradigms. This thesis aligns with that trend by explicitly separating tactical execution, itemization, and strategic planning into modular layers, each optimized for its respective temporal resolution and decision complexity.

## 2.2 Game AI and RL Applications in RTS/MOBA Games

The application of artificial intelligence to real-time strategy and multiplayer online battle arena games has served as a driving force for algorithmic innovation in machine learning and autonomous systems. Early game AI in RTS and MOBA domains relied heavily on hand-crafted rule sets, behavior trees, finite state machines, and goal-oriented action planning (GOAP). These systems were highly interpretable, computationally efficient, and easily debuggable, but they struggled to adapt to novel strategies, scale to complex decision spaces, or generalize across diverse game states. The introduction of Monte Carlo Tree Search (MCTS) and statistical learning techniques improved strategic foresight, but the real-time constraints and partial observability of MOBA games limited their direct applicability (??).

The paradigm shift occurred with the integration of deep reinforcement learning, particularly following breakthroughs in Atari, Go, and StarCraft. In the MOBA domain, OpenAI Five represented a landmark achievement, demonstrating that a team of five deep RL agents could defeat professional human players in Dota 2 through large-scale self-play, population-based training, and reward shaping (Berner et al., 2019). The system utilized a centralized critic with decentralized

actors, temporal abstraction via macro-actions, and a carefully engineered reward function that balanced combat, economy, and objective control. Similarly, DeepMind’s AlphaStar achieved grandmaster-level performance in StarCraft II by combining imitation learning from human replays, league-based self-play, and multi-agent coordination mechanisms (Vinyals et al., 2019). Both systems validated the potential of end-to-end neural policies in highly complex, partially observable environments.

However, these successes also exposed significant limitations that constrain broader adoption. First, the computational requirements are prohibitive for most academic and independent research settings. OpenAI Five trained on thousands of GPUs for months, while AlphaStar utilized massive TPU clusters and distributed simulation infrastructure. Second, end-to-end policies are inherently opaque, making it difficult to diagnose failures, interpret decision logic, or enforce safety constraints. Third, the reward engineering required to align neural policies with human-like strategic reasoning is highly non-trivial and often results in exploitative or unnatural playstyles. Finally, transferring knowledge across different heroes, maps, or game versions remains challenging due to the high sensitivity of flat policies to environmental distribution shifts (??).

Alternative approaches have sought to mitigate these issues by incorporating structural priors and modular design. Behavior tree-based systems remain widely used in commercial game AI due to their transparency, modularity, and ease of integration with existing engines (?). Hybrid architectures that combine symbolic planning with neural execution have shown promise in domains requiring both strategic reasoning and precise control (?). In Dota 2 specifically, several academic projects have explored imitation learning from replay data, supervised policy distillation, and curriculum learning to reduce sample complexity(?). These works emphasize the importance of leveraging domain structure rather than attempting to learn everything from scratch.

The literature thus points to a clear gap: while end-to-end RL can achieve elite performance, it does so at the cost of interpretability, reproducibility, and resource efficiency. Conversely, classical game AI systems are transparent and efficient but lack adaptive learning capabilities. A hybrid hierarchical framework that assigns each decision layer the most appropriate computational paradigm addresses this dichotomy. By delegating low-latency micro-control to deterministic controllers, medium-horizon economic optimization to offline RL, and high-level strategic reasoning to language-based planners, the proposed architecture aligns with emerging best practices in scalable, interpretable game AI.

## 2.3 Offline Reinforcement Learning for Itemization

Itemization in Dota 2 represents a sequential decision-making problem where agents must select purchases that optimize combat effectiveness, survivability, and role fulfillment under budget constraints and evolving game states. Traditionally, item builds have been treated as static guides or heuristic rules, but modern meta-analysis and data-driven approaches recognize itemization as a dynamic policy that responds to enemy composition, lane dynamics, timing windows, and economic trajectory. Reinforcement learning offers a principled framework for learning item-buying policies, but online exploration in a live match is computationally expensive, safety-critical, and often suboptimal due to sparse rewards and delayed feedback. Offline reinforcement learning has emerged as a compelling alternative by enabling policy optimization from fixed datasets of

historical interactions without further environment exploration (?).

The offline RL paradigm addresses several limitations of online RL in game environments. First, it eliminates the need for costly simulation infrastructure by leveraging existing replay datasets, which are abundant in MOBA ecosystems through platforms like OpenDota and Stratz. Second, it avoids catastrophic exploration, ensuring that the agent does not experiment with economically ruinous or tactically incoherent item sequences during training. Third, it enables policy evaluation and refinement through conservative value estimation and distributional matching techniques, which are critical when learning from static datasets that may contain suboptimal or noisy trajectories (?).

Foundational algorithms in offline RL include Behavior Cloning (BC), which learns policies via supervised imitation of expert demonstrations, and Deep Q-learning from Demonstrations (DQfD), which combines demonstration replay with temporal-difference updates to improve sample efficiency and policy quality (??). While BC is simple and stable, it suffers from compounding errors and inability to generalize beyond the demonstration distribution. DQfD mitigates this by integrating Q-learning dynamics, allowing the agent to refine its value estimates through bootstrapping while remaining anchored to expert data. However, both methods remain vulnerable to distributional shift when the learned policy encounters states not well-covered by the dataset.

To address extrapolation error, Conservative Q-Learning (CQL) introduces a regularization term that penalizes Q-values for out-of-distribution actions while preserving accurate estimates for in-distribution actions (?). This conservative update rule prevents the policy from overestimating the value of untested item sequences, a critical safeguard in economic decision-making where poor purchases can cascade into irreversible disadvantages. Subsequent methods such as TD3+BC and Implicit Q-Learning (IQL) further refine this balance by decoupling policy improvement from value function updates, using expectile regression, or applying implicit constraints on action support. These algorithms have demonstrated strong performance in robotic control, autonomous driving, and resource allocation tasks, where safety and data efficiency are paramount. In the context of Dota 2 itemization, offline RL is particularly well-suited for several reasons. First, replay datasets naturally capture expert trajectories, including meta-adaptive builds, situational pivots, and role-specific optimizations. Second, item purchases occur at discrete intervals with clear economic constraints, making them amenable to discrete or hybrid action space formulations. Third, the delayed impact of item choices (e.g., a defensive item purchased at 15 minutes influencing survivability at 30 minutes) aligns with the credit assignment challenges that offline RL algorithms are designed to handle. Recent studies have applied offline RL to character progression, loadout optimization, and skill sequencing in MOBA and RPG environments, demonstrating improved economic efficiency and meta-adaptation compared to static heuristics (??).

Despite these advantages, offline RL for itemization faces several practical challenges. Dataset quality varies significantly across patches, hero roles, and skill brackets, requiring careful preprocessing, filtering, and state representation engineering. Reward shaping must balance immediate gold efficiency with long-term strategic impact, avoiding myopic optimization that neglects power spikes or team synergy. Finally, integrating offline itemization policies with real-time tactical controllers requires careful latency management, fallback mechanisms, and consistency checks to prevent conflicting directives. This thesis addresses these challenges by employing a

structured data pipeline, conservative policy updates, and explicit orchestration logic that ensures safe, coherent item-buying behavior within the broader hierarchical framework.

## 2.4 Language Models and Agent Frameworks

The rapid advancement of large language models (LLMs) has catalyzed a paradigm shift in how autonomous agents handle high-level reasoning, planning, and environmental interaction. Unlike traditional reinforcement learning agents that map states to actions through numerical value functions or policy gradients, LLMs operate over semantic representations, enabling abstract reasoning, contextual memory, and tool-augmented decision-making. This capability has been leveraged to design agents that can interpret complex instructions, decompose tasks into subgoals, and dynamically adapt their behavior through iterative refinement. In game AI, LLMs offer a promising avenue for strategic planning, particularly in domains where high-level directives are naturally expressible in linguistic or symbolic form.

Early applications of LLMs in interactive environments focused on text-based games, narrative generation, and dialogue systems. However, recent work has demonstrated their potential in embodied and real-time decision-making through structured prompting, tool integration, and memory-augmented architectures. ReAct introduced a paradigm where LLMs interleave reasoning traces with actionable outputs, enabling them to decompose complex tasks, query external tools, and correct course based on feedback (?). This approach significantly improved task completion rates in interactive environments by grounding abstract reasoning in observable state changes. Toolformer extended this concept by enabling language models to self-learn tool usage through fine-tuning on API call demonstrations, reducing the need for explicit prompt engineering or external control loops (?).

In open-ended and dynamic environments, Voyager demonstrated the feasibility of LLM-driven agents that autonomously explore, learn skills, and construct executable code to interact with complex simulation worlds (?). By combining iterative prompting, skill libraries, and self-reflection, Voyager agents achieved unprecedented levels of adaptability and long-term progression without human intervention. Similarly, Generative Agents simulated human-like behavior in virtual environments by maintaining memory streams, planning routines, and social interactions through LLM-based cognition(?). These works collectively illustrate that LLMs can function as high-level cognitive engines, capable of abstracting over raw state representations, maintaining contextual continuity, and generating structured directives.

Applying LLMs to real-time, partially observable games like Dota 2 introduces unique engineering and algorithmic challenges. First, latency constraints demand efficient inference pipelines, as strategic decisions must be computed within tight temporal windows without disrupting tactical execution. Second, hallucination and overgeneralization pose risks in adversarial environments where incorrect strategic directives can lead to catastrophic resource misallocation. Third, grounding LLM outputs in game mechanics requires careful state abstraction, tool design, and validation layers to ensure that generated directives are executable and contextually appropriate. Recent studies have addressed these issues through constrained action spaces, structured JSON outputs, verification modules, and fallback heuristics, demonstrating that LLMs can reliably guide high-level behavior in complex games.



The integration of LLMs with hierarchical decision-making aligns with broader trends in neuro-symbolic AI and modular agent design. Rather than replacing learned or rule-based controllers, LLMs serve as meta-planners that synthesize game context, role objectives, and temporal constraints into actionable high-level directives. This division of labor leverages the LLM’s strength in semantic reasoning and contextual adaptation while delegating low-level execution to specialized controllers optimized for speed and reliability. Tool-augmented patterns further enhance this architecture by enabling the LLM to query game state, retrieve historical patterns, and issue structured commands to downstream modules. By framing strategic planning as a language-mediated orchestration problem, this approach bridges the gap between human-like reasoning and machine-executable control, offering a scalable pathway for adaptive game AI.

## 2.5 Evaluation Metrics in Game AI

Evaluating autonomous agents in complex, multi-objective environments like Dota 2 requires metrics that extend beyond traditional reinforcement learning benchmarks. While episode reward and win rate provide high-level performance indicators, they fail to capture the nuanced trade-offs inherent in MOBA gameplay, such as economic efficiency, tactical execution quality, role adherence, and strategic coherence. Consequently, the literature has developed a multi-faceted evaluation framework that combines quantitative performance metrics, behavioral analysis, and human-alignment assessments.

Win rate remains the primary objective metric, reflecting the agent’s ability to achieve the game’s core win condition against varied opponents. However, win rate alone is insufficient for diagnosing systemic weaknesses, as it aggregates over diverse match conditions and may mask inefficiencies in specific phases of gameplay. Complementary metrics include Kill-Death-Assist ratio (KDA), which measures combat effectiveness and positioning quality; Gold Per Minute (GPM) and Experience Per Minute (XPM), which quantify economic and progression efficiency; and objective control rates, which assess strategic prioritization of towers, Roshan, and map vision. These metrics collectively provide a granular view of agent performance across tactical, economic, and strategic dimensions.

Itemization evaluation presents additional challenges, as optimal builds are highly context-dependent. Traditional approaches rely on static build guides, but modern evaluation frameworks compare agent purchases against expert datasets using sequence similarity metrics, timing alignment, and role-specific optimization scores. Studies have employed dynamic time warping, edit distance, and probabilistic matching to quantify how closely learned item sequences align with professional trajectories, while also accounting for situational deviations and meta shifts. These methods enable researchers to distinguish between myopic gold spending and strategically coherent economic planning.

Strategic adherence and behavioral consistency are increasingly recognized as critical evaluation dimensions, particularly for hybrid and LLM-augmented agents. Metrics such as directive compliance rate, lane positioning accuracy, rotation timing, and objective synchronization measure how effectively high-level plans translate into in-game behavior. Ablation studies that isolate specific modules (e.g., disabling LLM tool-use or varying RL algorithms) provide causal insights into component contributions and failure modes. Furthermore, human-in-the-loop evaluations and

expert reviews offer qualitative assessments of playstyle naturalness, adaptability, and strategic depth, which are difficult to quantify but essential for real-world applicability.

Benchmarking frameworks in MOBA AI have evolved to address reproducibility and fairness. Controlled environments with standardized map seeds, fixed opponent pools, and deterministic randomization enable statistically significant comparisons across architectural variants. Population-based evaluation and cross-patch testing assess generalization capabilities, while role-stratified analysis ensures that performance metrics account for hero-specific responsibilities and team dynamics (?). The integration of these evaluation protocols into a cohesive testing pipeline is essential for validating hierarchical frameworks, as it reveals how well different layers interact under stress, latency constraints, and adversarial conditions.

Despite these advances, evaluation in game AI remains an open research area. Metrics must balance quantitative rigor with ecological validity, ensuring that laboratory performance translates to dynamic, human-like gameplay. The proposed framework addresses this by implementing a comprehensive evaluation suite that combines win-rate baselines, economic efficiency scores, itemization alignment metrics, and strategic adherence tracking, all analyzed through ablation and robustness testing. This multi-dimensional approach not only validates the hybrid architecture but also contributes to the broader development of standardized, interpretable evaluation methodologies for complex game AI systems.

## Chapter 3

# Methodology

The methodology of this research is grounded in a structured, iterative design process that bridges theoretical insights from hierarchical reinforcement learning, offline policy optimization, and large language model reasoning with the practical constraints of real-time game AI development. This chapter details the methodological workflow, beginning with a synthesis of existing literature to identify architectural gaps and justify design choices. It then outlines the prototyping and API integration workflow, followed by a comprehensive description of the three-level system architecture. The chapter further elaborates on the data pipeline construction, baseline formulation, and methodological validation protocols that ensure reproducibility, safety, and empirical rigor. By documenting each phase systematically, this methodology establishes a transparent pathway from conceptual design to functional implementation.

### 3.1 Literature Synthesis and Methodological Gaps

The initial phase of the research involved a systematic review of hierarchical decision-making frameworks, offline reinforcement learning algorithms, and language-model-driven agent architectures. The synthesis process followed a structured literature mapping approach, prioritizing peer-reviewed publications, conference proceedings, and validated technical reports published between 2015 and 2025. Key search domains included temporal abstraction in reinforcement learning, behavior tree implementations in commercial and academic game AI, offline policy learning from demonstration datasets, and tool-augmented reasoning in large language models. The review was conducted using academic databases such as IEEE Xplore, ACM Digital Library, arXiv, and SpringerLink, with keyword combinations tailored to MOBA AI, hierarchical control, and neuro-symbolic agent design.

The literature synthesis revealed three critical methodological gaps that directly informed the architectural design. First, while hierarchical reinforcement learning provides a rigorous mathematical foundation for temporal abstraction, most implementations struggle with boundary definition in dynamic, partially observable environments. Traditional option frameworks and MAXQ decompositions assume well-defined subtask initiation and termination conditions, which are rarely explicit in MOBA gameplay where strategic transitions are fluid and context-dependent (???). This gap necessitated a design that replaces rigid hierarchical boundaries with loosely coupled, asynchronously communicating modules, each operating on its own temporal clock while maintaining explicit fallback and arbitration mechanisms.

Second, offline reinforcement learning algorithms demonstrate strong sample efficiency and

safety guarantees when applied to static datasets, but their deployment in resource-constrained, real-time environments requires careful reward shaping and state representation engineering. Methods such as CQL and DQfD mitigate extrapolation error and distributional shift, yet they remain sensitive to dataset quality, role-specific variations, and patch-induced meta changes (????). The methodology therefore emphasizes a role-conditioned, feature-engineered state space and conservative policy updates that prioritize economic stability over aggressive optimization. This aligns with recent findings that offline RL performs best when constrained to well-covered regions of the state-action space and validated against domain-specific safety metrics (??).

Third, large language models exhibit remarkable capabilities in abstract reasoning and tool use, but their integration into real-time, latency-sensitive loops requires structured grounding mechanisms to prevent hallucination, output drift, and execution conflicts. While frameworks like ReAct and Voyager demonstrate the viability of iterative reasoning and self-correction, they operate in environments with generous computational budgets and minimal timing constraints (??). In Dota 2, strategic directives must be generated, validated, and dispatched within strict temporal windows without disrupting low-level tactical execution. This constraint motivated a methodology that treats the LLM as a high-level director rather than a direct controller, employing structured JSON outputs, tool-mediated state queries, and explicit latency budgets to ensure reliable orchestration.

By mapping these gaps against the requirements of a practical academic Dota 2 agent, the methodology crystallized around a hybrid hierarchical paradigm. Each layer was assigned a computational paradigm best suited to its temporal resolution and decision complexity: deterministic behavior trees for tactical execution, conservative offline RL for itemization, and tool-augmented language models for strategic planning. The integration of these layers was formalized through a Rust-based orchestration runtime designed to manage inter-module communication, enforce safety constraints, and handle fallback arbitration. This methodological framework ensures that theoretical insights are translated into a reproducible, empirically evaluable system that respects both algorithmic principles and engineering realities.

## 3.2 API Integration and Prototyping Workflow

The development of the agent framework required extensive interaction with Valve’s official Dota Bot Scripting API, which provides server-side Lua scripting capabilities for autonomous unit control (?). The prototyping phase followed an iterative, validation-driven workflow designed to understand API constraints, establish reliable state observation pipelines, and verify command execution latency before integrating higher-level learning and reasoning components.

Initial API exploration focused on state accessibility, command authorization, and timing constraints. The Dota bot interface exposes game-state information through a structured Lua API, including hero attributes, inventory, ability cooldowns, unit positions, lane assignments, and objective statuses. However, access is strictly bounded by in-game visibility rules: units hidden by the fog of war or cloaking abilities cannot be queried, and command issuance is limited to hero-controlled units or directly assigned creeps. Early prototyping validated these constraints by implementing a sandbox observation loop that recorded state deltas at fixed intervals, confirming

that partial observability must be explicitly modeled rather than circumvented. Command latency testing revealed that the API enforces a minimum execution interval for certain actions (e.g., ability casting, item activation, and purchase), necessitating a queuing mechanism that respects server-side timing gates.

To validate tactical behavior, a modular Lua sandbox was developed using behavior-tree-like control flows. The sandbox executed discrete decision nodes for movement, attack prioritization, ability usage, and retreat conditions. Each node was instrumented with logging hooks to record execution time, success rate, and state context. Validation matches were conducted against scripted bot opponents across varying difficulty tiers, with performance metrics tracked to ensure baseline competence before integrating learned or language-based components. The prototyping phase confirmed that low-latency tactical execution requires stateless, deterministic control flows that avoid heavy computation during active match loops. This insight directly informed the architectural decision to isolate tactical control from higher-level reasoning modules, ensuring that micro-decisions remain responsive under all conditions.

Parallel to tactical prototyping, a data pipeline was constructed to extract and preprocess replay data for offline reinforcement learning. The pipeline leveraged the OpenDota API (?) and community-maintained replay parsers to download match histories, parse combat logs, item purchase timestamps, gold trajectories, and hero progression curves. Raw replay data was filtered to exclude matches with anomalous duration, disconnected players, or non-standard game modes. Role assignment was inferred using heuristic clustering based on gold share, experience distribution, and early-game positioning patterns, aligning with established role-identification methodologies (?). The cleaned dataset was structured into state-action-reward tuples, with state features normalized across patches and hero types to ensure distributional stability.

Baseline reinforcement learning experiments were conducted in a simulated environment that replayed historical trajectories without interacting with the live game engine. Behavior cloning, DQfD, and CQL were implemented using PyTorch, with policy updates evaluated against held-out validation sets. The prototyping phase revealed that naive behavior cloning suffered from compounding errors when the agent encountered states outside the demonstration distribution, while DQfD improved sample efficiency but required careful temperature scheduling for demonstration replay weighting. CQL demonstrated the most robust performance by penalizing out-of-distribution Q-values, though it required extended training epochs to converge on stable policy distributions. These baseline comparisons informed the final algorithmic selection and hyperparameter configuration for the item-buying module, ensuring that the learned policy remained economically sound and role-aligned without requiring online exploration.

### 3.3 Architectural Design of the Three-Level Framework

The core methodological contribution of this work is a three-level hierarchical architecture that decomposes Dota 2 decision-making into tactical control, item-buying optimization, and strategic planning. Each layer operates on a distinct temporal resolution, employs a specialized computational paradigm, and communicates through a standardized orchestration protocol. The architecture was designed to satisfy four methodological requirements: modularity, interpretability, latency compliance, and safety enforcement.

### 3.3.1 Tactical Layer: Behavior-Tree Execution and Micro-Control

The tactical layer is responsible for low-frequency mechanical execution, including movement, attack targeting, ability casting, positioning, and immediate threat response. This layer operates at a fixed tick rate synchronized with the game engine’s update loop, ensuring that commands are issued within the API’s latency budget. The implementation relies on a behavior-tree-like control flow architecture, which structures decision logic as a hierarchical graph of nodes evaluating boolean conditions and executing atomic actions (?).

Behavior trees were selected over finite state machines or neural micro-policies due to their explicit execution semantics, debuggability, and resistance to catastrophic failure modes. The tree is organized into composite nodes (selectors, sequences, and parallel decorators), condition nodes (state evaluators for cooldowns, health thresholds, and enemy proximity), and action nodes (movement commands, ability casts, and item activations). Each node maintains local context but does not store long-term memory, ensuring that tactical decisions remain responsive to real-time state changes.

Methodologically, the tactical controller was designed with explicit fallback pathways. If a condition node fails or an action node encounters a server-side timing constraint, the tree backtracks to the nearest valid selector, ensuring graceful degradation rather than execution stalls. Priority arbitration is enforced through node ordering, with survival-critical behaviors (e.g., retreat on low health, ability cancellation on silence) positioned at the highest priority levels. This design aligns with real-time control theory, which emphasizes deterministic execution paths and bounded latency in safety-critical systems (?). The tactical layer receives high-level directives from the orchestration runtime but retains autonomy over micro-execution, ensuring that strategic suggestions never override immediate survival requirements.

### 3.3.2 Item-Buying Layer: Offline Reinforcement Learning Formulation

The item-buying layer addresses medium-horizon economic optimization, learning purchase policies from historical replay data without online exploration. This layer is formulated as a discrete-action offline reinforcement learning problem, where the state space encodes hero attributes, current inventory, gold balance, game time, lane status, and enemy composition features. The action space consists of a curated subset of purchasable items, filtered for role relevance and meta viability.

The methodological design emphasizes conservative policy updates and distributional safety. The reward function is constructed as a weighted combination of gold efficiency, timing alignment with power spikes, role-specific objective contribution, and win-condition proxy signals. To prevent myopic optimization, delayed reward shaping incorporates exponential discounting aligned with typical match durations, while role-conditioned baselines ensure that support and core heroes optimize for different economic trajectories (??).

Policy training follows a two-stage methodology. First, behavior cloning initializes the policy network using supervised imitation of expert trajectories, providing a stable foundation anchored to human-like purchase patterns. Second, DQfD and CQL refinements iteratively adjust Q-value estimates using temporal-difference updates and conservative regularization. The CQL loss function penalizes Q-values for actions outside the demonstration distribution, preventing the policy from overestimating untested item sequences 3.1:

$$\mathcal{L}_{\text{CQL}} = \mathbb{E}_{s \sim \mathcal{D}} \left[ \log \sum_a \exp(Q(s, a)) - Q(s, \pi(s)) \right] \quad (3.1)$$

where  $\mathcal{D}$  represents the replay dataset,  $Q$  denotes the value network, and  $\pi$  is the learned policy. This formulation ensures that the item-buying layer remains economically coherent even when encountering novel game states, as conservative updates restrict policy exploration to well-covered regions of the state space (??).

Methodologically, the item-buying module operates asynchronously from the tactical loop, issuing purchase requests to the orchestration runtime at predefined gold thresholds or item-shop proximity events. The runtime validates each request against current gold balance, inventory slots, and role constraints before dispatching the command to the Lua controller. This decoupled execution model prevents economic decisions from disrupting tactical responsiveness while maintaining explicit safety checks.

### 3.3.3 Strategic Layer: Language-Model Reasoning and Tool Integration

The strategic layer functions as a high-level planning module, generating contextual directives for lane pressure, objective timing, rotation suggestions, and team posture. This layer leverages a large language model configured for tool-augmented reasoning, structured output generation, and iterative state synthesis. Unlike end-to-end neural policies, the LLM operates over semantic abstractions, enabling interpretable strategic adaptation without requiring massive-scale training infrastructure.

The methodology employs a ReAct-inspired reasoning loop, where the LLM interleaves contextual analysis, tool queries, and directive generation (?). The model receives a structured prompt containing hero role description, current match phase, gold/experience differentials, objective statuses, and recent combat logs. It then executes a constrained reasoning cycle: (1) analyze state context, (2) query external tools for missing information (e.g., enemy last-known positions, objective respawn timers), (3) synthesize strategic directive, and (4) output structured JSON for downstream consumption.

Tool integration is methodologically grounded in explicit function definitions and output validation. The LLM is provided with a set of callable tools implemented in the orchestration runtime, including `query_lane_status`, `check_objective_timer`, `estimate_enemy_rotation`, and `recommend_team_posture`. Each tool returns validated responses that prevent hallucinated state information. The LLM's output is constrained to a predefined JSON schema for phase transitions (e.g., `laning_to_mid_game`, `objectives_pawns`, `item_powers_pikes`). Between invocations, the LLM maintains state control loops. The methodological design ensures that strategic reasoning remains grounded, executable, and deterministic.

### 3.3.4 Orchestration Runtime: Cross-Module Communication and Safety Arbitration

The orchestration runtime serves as the methodological backbone of the framework, managing inter-module communication, enforcing safety constraints, and arbitrating conflicting directives. Implemented in Rust for memory safety, deterministic execution, and low-latency performance, the runtime acts as a centralized message bus that coordinates the Lua tactical controller, Python-based item-buying RL module, and external LLM inference service.

The communication architecture follows a publish-subscribe model with explicit priority queues. Modules publish state updates and action requests to the runtime, which routes messages

based on temporal urgency and dependency constraints. Tactical commands are routed through a high-priority queue with sub-millisecond latency guarantees, item-buying requests traverse a medium-priority queue with validation checkpoints, and strategic directives flow through a low-priority queue with asynchronous acknowledgment. This tiered routing ensures that micro-control remains uninterrupted while higher-level planning executes in the background.

Safety arbitration is enforced through a multi-layered validation pipeline. First, syntactic validation ensures that all messages conform to predefined schemas. Second, semantic validation checks for logical consistency (e.g., item purchases require sufficient gold, tactical retreats override aggressive positioning directives). Third, priority arbitration resolves conflicts using a deterministic rule set: survival commands > role-preserving economic actions > strategic suggestions. If a module fails to respond or produces malformed output, the runtime triggers fallback mechanisms, reverting to conservative heuristics until stability is restored.

The runtime also implements explicit latency budgeting and timeout handling. Each module is allocated a maximum execution window; if exceeded, the runtime either caches the last valid state or initiates a safe degradation protocol. Inter-process communication (IPC) is optimized through shared memory segments and zero-copy serialization where possible, minimizing overhead while maintaining strict isolation boundaries. This methodological design ensures that heterogeneous components operate cohesively under real-time constraints, a critical requirement for deployable game AI systems (?).

### 3.4 Data Pipeline and Baseline Construction

A rigorous data pipeline was constructed to ensure that the item-buying reinforcement learning module trains on high-quality, role-conditioned, and patch-consistent datasets. The pipeline methodology encompasses data acquisition, parsing, filtering, feature engineering, reward shaping, and baseline validation.

Data acquisition leveraged the OpenDota API and community replay archives, downloading matches from professional tournaments and high-skill bracket ladders. Matches were filtered to exclude custom games, short-duration anomalies, and non-standard rule sets. Replay parsing utilized open-source parsers to extract tick-by-tick state logs, including hero positions, item purchases, gold flows, ability usage, and combat outcomes. The raw logs were structured into episode trajectories aligned with role assignments, using heuristic clustering validated against established role-identification frameworks (?).

Feature engineering transformed raw logs into normalized state representations. Continuous variables (gold, experience, health, mana) were scaled to zero-mean, unit-variance distributions. Categorical variables (hero type, item slot occupancy, lane assignment) were one-hot encoded or embedded via lightweight neural projections. Temporal features (game time, objective respawn windows, recent combat frequency) were encoded as sinusoidal positional embeddings to preserve periodic structure. The state representation was designed to be patch-invariant where possible, using relative gold differentials and role-normalized progression metrics rather than absolute values.

Reward shaping followed a multi-objective formulation. Primary rewards aligned with gold efficiency and timing optimization, penalizing wasted purchases or delayed power spikes. Secondary rewards incorporated role-specific objectives, such as vision placement for supports or



farm prioritization for cores. Tertiary rewards provided sparse win-condition signals, ensuring that economic optimization remained aligned with long-term strategic outcomes. The reward weights were calibrated through ablation testing to prevent dominance of any single objective.

Baseline construction involved training three policy variants on the processed dataset: pure behavior cloning, DQfD with demonstration replay, and CQL with conservative regularization. Each variant was evaluated on held-out validation matches using metrics such as item-build similarity, timing alignment, gold efficiency, and win-condition correlation. The CQL variant demonstrated the most robust performance, achieving higher economic coherence and fewer distributional violations. This baseline comparison informed the final policy selection and hyperparameter configuration, ensuring methodological transparency and reproducibility.

### 3.5 Methodological Constraints and Validation Framework

The methodology explicitly acknowledges and addresses several constraints inherent to real-time game AI research. First, partial observability is handled through state estimation heuristics and tool-mediated LLM queries rather than full-state assumption. The tactical layer relies only on visible information, while the strategic layer incorporates probabilistic enemy position estimates based on last-known sightings and lane pressure patterns. This design aligns with information-constrained decision-making principles, ensuring that the agent operates under realistic game conditions (Berner et al., 2019).

Second, non-stationarity induced by meta shifts and patch updates is mitigated through role-conditioned state normalization and conservative policy updates. The item-buying RL module avoids overfitting to specific patch distributions by training on multi-patch datasets and applying distributional regularization. Strategic directives are formulated as flexible recommendations rather than rigid scripts, allowing adaptive interpretation by the tactical layer.

Third, reproducibility is ensured through deterministic seed initialization, standardized opponent pools, and controlled match configurations. The orchestration runtime logs all module interactions, state transitions, and directive executions, enabling post-hoc analysis and independent validation. Experimental protocols follow a stratified design, testing across multiple heroes, roles, and difficulty tiers to assess generalization capabilities.

The validation framework combines quantitative metrics with qualitative behavioral analysis. Quantitative evaluation includes win rate, KDA, GPM/XPM, item-build similarity, and objective timing accuracy. Qualitative analysis involves expert review of strategic coherence, tactical responsiveness, and economic decision quality. Ablation studies isolate individual modules to assess their contribution to overall performance, while robustness testing evaluates system stability under latency variations, state corruption, and adversarial conditions.

By documenting each methodological step with explicit design rationale, constraint handling, and validation protocols, this chapter establishes a rigorous foundation for the implementation, experimentation, and evaluation phases detailed in subsequent chapters. The methodology ensures that the hybrid hierarchical framework is not only theoretically grounded but also empirically verifiable, reproducibly implementable, and practically deployable within academic research constraints.

## Chapter 4

# Frequently asked questions

In addition to the information provided in chapter 1, here are some brief notes on references (see section 4.1) and figures (see section 4.2).

### 4.1 References

You can, of course, use any referencing style you like such as plain. The natbib package, however, allows you to do this with named style citations:

<code>\citet{key}</code>	Jones et al. (1990)
<code>\citet*{key}</code>	Jones, Baker, and Smith (1990)
<code>\citep{key}</code>	(Jones et al., 1990)
<code>\citep*{key}</code>	(Jones, Baker, and Smith, 1990)
<code>\citep[chap. 2]{key}</code>	(Jones et al., 1990, chap. 2)
<code>\citep[e.g.][] {key}</code>	(e.g. Jones et al., 1990)
<code>\citep[e.g.] [p. 32] {key}</code>	(e.g. Jones et al., p. 32)
<code>\citeauthor{key}</code>	Jones et al.
<code>\citeauthor*{key}</code>	Jones, Baker, and Smith
<code>\citeyear{key}</code>	1990

This template uses BibTeX as the citation engine, and you should learn to fill in citations correctly, perhaps even use a bibfile manager like JabRef. Recall that when citing a book, you should always include the chapter, for example, the first chapter of the classic AI book (Russell and Norvig, 2022, Ch. 1). Papers are easier to cite, such as a recent AAAI paper Amado et al. (2022).

### 4.2 Figures

To include an encapsulated postscript or PDF file (depending on whether you're using L<sup>A</sup>T<sub>E</sub>X or PDFL<sup>A</sup>T<sub>E</sub>X) as a figure, do something like the following. Note, to ensure correct cross-referencing, it is best to include the figure label within the caption definition. *Note that the graphicx package is already loaded and used to include the University crest on the title page.*

```
\begin{figure}
  \begin{center}
    \includegraphics{myfigure.pdf}
```

```

\caption{This is my figure.\label{fig:mylabel}}
\end{center}
\end{figure}

```

### 4.3 Frequently used symbols

In  $\text{\LaTeX}$  documents where you want to use a modality or some text consistently in normal text and in equation environments it is often difficult to remember to typeset the text consistently or time-consuming to keep typing in the environment. It may be a good idea to define something like the following in the preamble (i.e. before `\begin{document}`):

```

\def\sftthing#1#2{\def#1{\mbox{{\small\normalfont\sffamily #2}}}}

\sftthing{\PP}{P}
\sftthing{\FF}{F}

```

Then use it in text or math mode. In all cases it looks the same; e.g.  
`\PP\` refers to something, and other things are `\FF`;  $\Phi = \PP \cup \FF$   
 is typeset as:

$P$  refers to something, and other things are  $F$ ; i.e.  $\Phi = P \cup F$

Note that you need to put “`\`” after the command if you want a normal space after it.

# Bibliography

- Amado, L. R., Mirsky, R., and Meneguzzi, F. (2022). Goal Recognition as Reinforcement Learning. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Press.
- Berner, C., Brockman, G., Chan, B., Cheung, V., Dkebiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.
- Russell, S. J. and Norvig, P. (2022). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 4th edition.
- Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575:350–354.